HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

Bachelorarbeit

Herr

Lutz Groß

Entwicklung einer Zustandsanzeige heterogener Testsysteme

Mittweida, 2012

Fakultät Elektro- und Informationstechnik

Bachelorarbeit

Entwicklung einer Zustandsanzeige heterogener Testsysteme

Autor:

Herr

Lutz Groß

Studiengang:

Informationstechnik

Seminargruppe:

IT07wk-B

Erstprüfer:

Professor Dr.-Ing. habil. Lutz Winkler

Zweitprüfer:

Dipl. Math. Karsten Rackwitz

Einreichung:

Mittweida, 15.03.2012

Verteidigung/Bewertung:

Mittweida, 2012

Bibliografische Angaben

Groß, Lutz:

Entwicklung einer Zustandsanzeige heterogener Testsysteme im Sinne der frühzeitigen Erkennung elementarer Fehlerquellen vor der Ausführung komplexerer Testszenarien - 2012 .

Mittweida, Hochschule Mittweida (FH), University of Applied Sciences,

Fakultät Elektro- und Informationstechnik, 69 Seiten, 25 Abbildungen, 3 Tabellen, 2 Anlagen, Bachelorarbeit, 2012

Referat

Kommunikationsnetzwerke werden durch die ständig wachsende Anzahl neuer und komplexerer Hard- und Softwarekomponenten immer vielschichtiger. Die Verwaltung und Überwachung der Einzelkomponenten erfolgt über unterschiedlichste Zugriffsverfahren. Der Aufwand, diese Systeme vor dem Einsatz zu testen und kundenspezifisch einzurichten, steigt mit der Komplexität an.

Für die sinnvolle Auswertung solcher komplexer Testszenarien muss im Vorfeld gewährleistet sein, dass alle nötigen Netzelemente eines Szenarios funktionstüchtig sind und miteinander kommunizieren können. Ist dies nicht der Fall, entstehen Fehler während des Tests, die das Ergebnis verfälschen und unbrauchbar machen. In der vorliegenden Bachelorarbeit ist eine Anwendung entwickelt worden, die den aktuellen Zustand von einzelnen Netzelementen und kompletten Testszenarien auf einer Übersichtsseite darstellt. Jedem Mitarbeiter wird, mithilfe dieser Übersichtsseite, die Möglichkeit gegeben, vor der Durchführung eines komplexen Testfalles eventuelle andere Fehlerquellen zu erkennen und zu beseitigen.

Abstract

Due to the ever-growing number of new and complex hard- and software components, communication networks are becoming more and more complex. Managing and monitoring of the individual components is performed using different access methods. The effort to customer-specific set up and test these systems increases with the complexity.
To ensure a meaningful analysis of such complex test scenarios, it must be guaranteed in advance that all necessary network elements are fully functional and communicate with each other. If this is not the case, errors may occur during the test, corrupting the results and making them essentially useless.
During the course of this bachelor thesis, an application has been developed that represents the current state of the individual network elements and complete test scenarios on one page. It gives every employee the opportunity to recognize other possible sources of error and correct these before performing a complex test case.

Vorwort

Diese Bachelorarbeit wurde in der Abteilung für Mobilfunktestsysteme der Firma Nokia Siemens Networks in Berlin angefertigt. Wesentliche Aufgabe dieser Abteilung ist die Durchführung komplexer Tests im Bereich der Mobilfunktechnik und speziell dem IN[1]-System. Verschiedene IN-Services und damit unterschiedliche Protokolle werden dabei in vielfältigen Szenarien getestet. Wichtig dabei ist es, die kundenspezifische Konfiguration des Netzes möglichst nahe nachzubilden, um spätere Fehler im Einsatzgebiet zu reduzieren und im besten Falle auszuschließen.

Mein Dank gilt vor allem meinen Betreuern Herrn Dipl. Math. Karsten Rackwitz und Herrn Professor Dr. rer. nat. Sergej Alekseev, sowie Professor Prof. Dr.-Ing. habil. Winkler für die Unterstützung, die gezeigte Geduld und ihr Verständnis in allen guten und vor allem auch in den schwierigen Phasen meiner Abschlussarbeit.

Ein Besonderer Dank geht auch an alle Mitarbeiter der Arbeitsgruppe, die mir in vielen, teils hitzigen Diskussionsrunden, bei der Entwicklung und Durchführung meiner Abschlussarbeit geholfen haben.

[1] Intelligent Network

Inhalt

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

AMPS	Advanced Mobile Phone Service
ATM	Asynchronous Transfer Mode
AuC	Authentication Center
BSC	Base Station Controller
BTS	Base Transceiver Station
CAMEL	Customized Applications for Mobile Network Enhanced Logic
DECT	Digital Enhanced Cordless Telecommunications
DTMF	Dual-tone multi-frequency
EIR	Equipment Identity Register
EWSD	Elektronisches Wählsystem Digital
GGSN	Gateway GPRS Support Node
GPRS	General Packet Radio Service
GSM	Global System for Mobile Communications
HLR	Home Location Register
HSDPA	High Speed Downlink Packet Access
HSUPA	High Speed Uplink Packet Access
IMSI	International Mobile Subscriber Identity
IN	Intelligent Network
INAP	Intelligent Network Application Part
LDAP	Lightweight Directory Access Protocol
MAP	Mobile Application Part
MGW	Media Gateway
MOC	Mobile Originating Call
MSC	Mobile Switching Center
MSRN	Mobile Station Roaming Number
MTC	Mobile Terminating Call
NGMN	Next Generation Mobile Networks
RNC	Radio Network Controller

SCP	Service Control Point
SIP	Session Initiation Protocol
SMS	Small Message Service
SS7	Signalisierungssystem 7
UMTS	Universal Mobile Telecommunications System
USSD	Unstructured Supplementary Service Data
UTRAN	UMTS Terrestrial Radio Access Network

1 Einleitung

1.1 Zielsetzung

Bedingt durch die zunehmend komplexer werdenden Kommunikationsnetze steigt auch der notwendige Testaufwand dieser Systeme vor der Implementierung beim Kunden. Verschiedenste, immer umfangreicher werdende, Testfälle müssen durchgeführt und ausgewertet werden, um die Anforderungen an das System zu bestätigen. Je komplexer die Testfälle werden, umso wichtiger wird dabei, dass die grundlegenden Funktionen der einzelnen Komponenten einwandfrei und wie vorgesehen funktionieren. Oft kommt es in komplexen Tests zu Fehlern, die auf Netzwerkstörungen zurückzuführen sind und nicht unbedingt etwas mit dem eigentlichen Testfall zu tun haben. Der Tester muss diese Fehler schnell erkennen und beheben können und kann erst dann den eigentlichen Testfall wiederholen. Das Auftreten solcher elementarer Fehler behindert den Tester in seiner eigentlichen Testarbeit und verlangsamt so den Prozess der Fehlerfindung und Korrektur.

Um die Auswirkungen zu minimieren wird in der vorliegenden Bachelorarbeit die Entwicklung einer Zustandsanzeige komplexer Testsysteme beschrieben. Mit dieser Anzeige soll dem Tester die Möglichkeit gegeben werden vor jedem komplexen Testfall prüfen zu können, ob sämtliche Hard- und Softwarekomponenten (im Folgenden auch „Netzelemente“ oder kurz: “NE“ genannt) des zu testenden Systems (im Folgenden „Projekt“ genannt) in seinen grundlegenden Funktionen verfügbar und funktionstüchtig sind. Außerdem soll die Möglichkeit gegeben werden den Gesamtzustand eines Projektes zu ermitteln, also die Kommunikation der Netzelemente untereinander zu überprüfen. Änderungen und Anpassungen sollen durch das Bedienpersonal schnell und unkompliziert vorgenommen werden können.

Zur Anwendungsentwicklung sind in der Arbeitsgruppe vorhandene Entwicklungsprogramme zu nutzen, um eine anschließende einfache Betreuung und Weiterentwicklung durch einen Mitarbeiter zu ermöglichen. Außerdem ist zu beachten, dass der Einsatzort ein Desktop-PC mit eingeschränkter Leistung sein wird, an dem ein Mitarbeiter seiner eigentlichen Arbeit nachgeht. Die Komplexität der Anwendung ist also auf ein Minimum zu reduzieren, um die Arbeitsfähigkeit des PC’s und somit des Mitarbeiters nicht unnötig einzuschränken.

1.2 Kapitelübersicht

Zu Beginn dieser Arbeit wurde im **ersten Kapitel** die gestellte Aufgabe analysiert und dabei die Vorteile und Ziele dieser Arbeit herausgearbeitet.

Nachdem die Aufgabe analysiert wurde, erfolgt im **Kapitel 2** ein kurzer Einblick in die Geschichte der Mobilfunkentwicklung bis zur 4. Generation. Dabei wird kurz auf die Funktionen und Aufgaben einzelner wichtiger Netzelemente eingegangen, sowie am Modell eines Mobile-Mobile-Calls der Verbindungsaufbau eines Rufes dargestellt und beschrieben. Im Anschluss wird das IN-System vorgestellt und am Ende des Kapitels der allgemeine Testablauf bei NSN erläutert.

Kapitel 3 stellt ausgewählte, bereits existente Softwarevarianten zur Überwachung und Visualisierung von Netzwerken vor. Die daraus gewonnenen Erkenntnisse werden zusammengefasst und Schlüsse für die eigene Lösung gezogen.

Im **Kapitel 4** werden die gesammelten Grundgedanken und Vorüberlegungen des Autors vertieft und daraus resultierend der Ansatz der Softwareentwicklung beschrieben.

Es folgt im **Kapitel 5** die Beschreibung der Entwicklung der Software in Einzelschritten.

Im Mittelpunkt des **Kapitels 6** steht die Zusammenfassung der erreichten Ergebnisse. Dabei wird auf aufgetretene Probleme und deren Lösung eingegangen. Am Ende dieses Kapitels erfolgt ein Ausblick auf mögliche Weiterentwicklungen der entwickelten Software.

2 Grundlagen

Im folgenden Kapitel werden Eckdaten der Entwicklung des Mobilfunks zusammengefasst dargelegt. Dabei wird hauptsächlich auf Deutschland Bezug genommen und anschließend die Entwicklung des länderübergreifenden Standards GSM[2] bis zur vierten Generation beschrieben. Im zweiten Teil dieses Kapitels erfolgen die Erläuterungen eines Verbindungsaufbaues zwischen zwei Mobilfunkteilnehmern und eine kurze Einführung in Intelligente Netze. Abschließend wird der Ablauf von Tests komplexer Mobilfunksysteme am Beispiel von NSN[3] kurz erläutert. Der zweite Teil dieses Kapitels dient hauptsächlich dazu, dem Leser ein allgemeines Verständnis zu vermitteln.

2.1 Mobilfunk Entwicklungsgeschichte (Elektronik Kompendium 2010)

Die Geschichte des Mobilfunks beginnt bereits 1879 als David Edward Hughes das Phänomen der elektromagnetischen Welle vorführt. Heinrich Rudolf Hertz reproduzierte ein paar Jahre später dieses Phänomen mithilfe eines Oszillators, der in einem sich in der Nähe befindlichen Empfänger eine Spannung erzeugte und bestätigte somit Hughes Beobachtungen. 1897 entwickelte Guglielmo Marconi das Morsen und somit das erste brauchbare System zur drahtlosen Übertragung von Nachrichten. Das erste Mobiltelefon fürs Auto gab es bereits 1935. Die nötige Technik nahm damals den gesamten Kofferraumplatz in Anspruch.

Die ersten Mobilfunknetze in Deutschland waren analog und entstanden auf der Basis des amerikanischen Standards AMPS[4]. 1952 wurde erstmals in Deutschland eine Telefonverbindung zwischen einem Mobilfunkteilnehmer und einem Festnetzteilnehmer geschaltet. Die eingesetzten Geräte konnten allerdings nur im Zusammenspiel mit dem entsprechenden Netz eingesetzt werden. Da in verschiedenen Städten unterschiedliche Funksysteme genutzt wurden war die Nutzung eines bestimmten mobilen Teilnehmers nicht beliebig in jeder Stadt möglich. Das erste landesweite Mobilfunknetz entstand 1958. Der Name dieses Netzes lautete A-Netz. Die Vermittlung musste noch per Hand vorgenommen werden, allerdings konnte sich nun jedes beliebige Funktelefon an jede Basisstation anmelden.

Die automatische Vermittlung wurde dann mit dem sogenannten B-Netz im Jahre 1972 in

[2] Global System for Mobile Communications
[3] Nokia Siemens Networks
[4] Advanced Mobile Phone Service

Deutschland, Österreich, Niederlande und Luxemburg eingeführt. Außerdem war es ab dem B-Netz möglich, länderübergreifend zu telefonieren (Roaming). Ein großer Nachteil dieses Netzes war es allerdings, dass man genau wissen musste, wo sich der andere Teilnehmer aufhielt. Über eine entsprechende Vorwahl konnte man sich zu der entsprechenden Funkzelle verbinden und ein Gespräch führen. Bedingt durch die wenigen Funkstationen die bis Dato existierten, war es außerdem meist schwer einen freien Funkkanal zu bekommen.

1986 wurde in Deutschland das C-Netz eingeführt. Zum Leistungsumfang gehörte nun die automatische Weitervermittlung (Handover) von Mobilteilnehmern zwischen den einzelnen Funkzellen einer Basisstation. Der Standort des Mobilteilnehmers wurde nun ständig aktualisiert (vollautomatische Mobilitätsverwaltung) und erlaubte somit die Vermittlung ohne manuelles eingreifen. In dieser Zeit entstanden in Europa viele verschiedene zueinander inkompatible Mobilfunksysteme, sodass das Handover, wie es aus dem B-Netz bereits bekannt war, nicht mehr funktionierte. Mit der Entwicklung des länderübergreifenden Standards „GSM – Global System for Mobile Communications“ wurde die 2. Generation von Mobilfunknetzen eingeläutet. Diese Entwicklungen führten zum Durchbruch der mobilen Sprachübertragung und ermöglichten die Ausbildung eines globalen digitalen Mobilfunknetzes.

2.1.1 2G Global System for Mobile Communications (GSM)

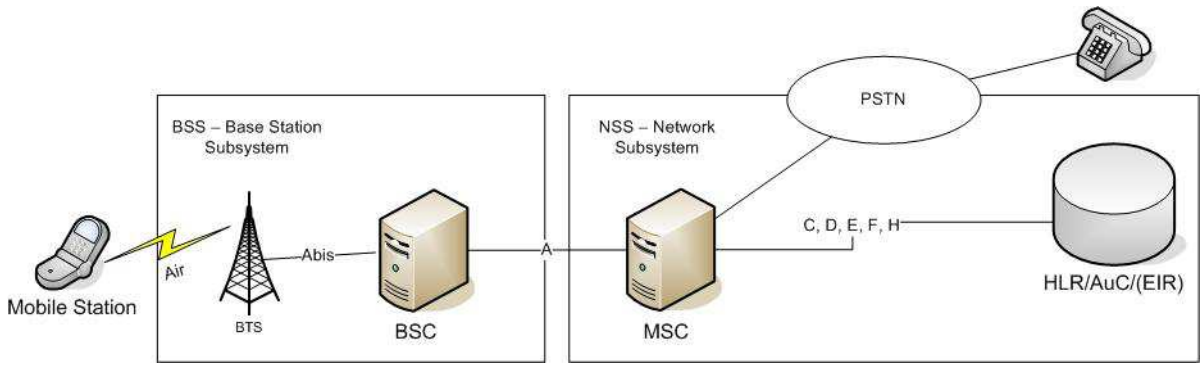

Abbildung 1: 2G GSM

In Abbildung 1 wird das Grundmodell des GSM-Netzes, als ein Standard der 2. Generation von Mobilfunknetzen, dargestellt. Der Vollständigkeit halber sei hier noch erwähnt, dass auch das D-Netz, das E-Netz und HSCSD[5] zu Standards der 2. Generation gehören. GSM ist dabei der meist verwendete Standard in Europa. Im Gegensatz zur 1. Generation erfolgt nun

[5] High Speed Circuit Switched Data

die Kommunikation komplett digital. Als Frequenzbereiche werden in Europa 900 Mhz und 1800 Mhz genutzt. In Amerika benutzt man 1900 Mhz. Die wichtigsten Elemente des GSM-Netzes der 2. Generation sind BTS[6], BSC[7], MSC[8], HLR[9], AuC[10] und EIR[11]. Die Kommunikation zwischen den einzelnen Elementen übernehmen verschiedenste Schnittstellen (in Abbildung 1 erkennbar: „A“-, „Abis“-,“C“-,“D“-,“E“-,“F“-,“H“-Interface).

Das gesamte GSM-Netz besteht aus einzelnen zusammengesetzten Funkzellen. Die Größe dieser Zellen hängt von dem verwendeten Frequenzband ab. Bei GSM 900 (900 Mhz) können Funkzellen einen Durchmesser von bis zu 35 km und bei GSM1800 (1800 Mhz) 8 km haben. Benachbarte Zellen arbeiten in unterschiedlichen Teilfrequenzbereichen, damit sie sich nicht gegenseitig stören. Befindet man sich mit einem GSM-fähigen Mobilteil im GSM-Netz, verbindet sich das Mobilteil automatisch mit der Basistation, die am nächsten und somit am signalstärksten ist (BTS). Jede BTS ist an einen BSC angeschlossen. Die BTS übernimmt dabei alle funktechnischen Aufgaben, wie die Verbindungsherstellung zwischen Mobilteil und Netz, sowie der Überwachung des Sende- und Empfangsverlaufes zum Mobilteil. Außerdem realisiert sie die Übertragung von Signalisierungsinformationen an den BSC, welcher seinerseits die Steuereinheit der BTS ist. Die Gesamtheit aus Basisstationen und Basisstationscontroller nennt man BSS[12] oder GERAN[13]. Die Schnittstelle zum NSS[14] bildet die Verbindung des BSC mit dem MSC. Es können mehrere BSC an eine MSC angeschlossen werden. Die Funktion eines MSC ähnelt der einer digitalen Vermittlungsstelle. Sie sind untereinander verbunden und bilden so ein Netzwerk. Über dieses Netz aus MSC’s werden Vermittlungen zwischen Mobilteilnehmern hergestellt. Durch die Anbindung an die Vermittlungsstelle des öffentlichen Festnetzes (PSTN - Public Switched Telephone Network) werden auch Verbindungen von Mobilteilen zu Festnetzteilnehmern und umgekehrt realisiert. Als Speicher für alle Nutzerinformationen zu allen Mobilfunkteilnehmern eines GSM-Netzes wird das HLR verwendet. Jedes GSM-Netz hat mindestens ein HLR. Zu den Nutzerinformationen gehören Telefonnummern, freigeschaltete benutzerspezifische Dienste sowie Informationen zum aktuellen VLR[15] eines jeden Mobilfunkteilnehmers. Das VLR dient der Entlastung des HLR. Hier werden temporär

[6] Base Transceiver Station
[7] Base Station Controller
[8] Mobile Switching Center
[9] Home Location Register
[10] Authentication Center
[11] Equipment Identity Register
[12] Base Station Subsystem
[13] GSM/EDGE Radio Access Network
[14] Network Subsystem
[15] Visitor Location Register

alle Nutzerinformationen der Mobilfunkteilnehmer hinterlegt, die sich im Einzugsgebiet der VLR befinden. Im HLR wird dabei vermerkt, in welcher VLR sich ein Teilnehmer befindet. Sobald sich dieser Mobilfunkteilnehmer aus dem Einzugsbereich entfernt, werden seine Daten aus dem VLR gelöscht und die Information der neuen VLR im HLR vermerkt.

Neben dem HLR und dem VLR gibt es noch weitere Datenbanken. Zum einen gibt es das EIR, indem alle zugelassenen IMEI[16] Nummern eines GSM-Netzes eingetragen sind. Jedes Mobilfunkendgerät hat eine eigene IMEI. Mithilfe des EIR können so Verbindungsversuche gesperrter Mobiler Endgeräte identifiziert und verhindert werden. Ein weiteres Datenbankelement ist das AuC. Hier werden u.a. die PIN[17] Nummern von SIM[18]-Karten gespeichert. EIR und AuC sind somit entscheidende Elemente, wenn sich ein Teilnehmer im GSM-Netz anmeldet (einschalten eines Mobilteilnehmers).

Die Nutzdatenrate bei GSM war zu Beginn auf 9,6 kbit/s beschränkt. Durch fortschrittlichere Kanalcodierung und Minimierung der Fehlerkorrektur erhielt man später eine Nutzdatenrate von 14,4 kbit/s bei allerdings höherer Fehleranfälligkeit.

Für digitale Daten mit kleinem Volumen, z. B. SMS, MMS, reichte das aus. Für aufwendigere Internet- und Multimediaanwendungen war dies allerdings unzureichend. Somit war GSM nur der erste Schritt auf dem Weg zum Mobilen Internet. (Wikipedia: GSM 2010)

Mit der Entwicklung von HSCSD und GPRS wurden schon sehr viel höhere Datenraten realisierbar. (Mobilfunk 2010)

2.1.2 2.5G General Packet Radio Service (GPRS)

Ein weiterer wesentlicher Schritt war die Entwicklung von GPRS. Man erreichte nun eine theoretische Übertragungsrate von bis zu 171 kbit/s. Dieser Standard ermöglichte erstmalig paketorientiert Daten zu übertragen. Die Übertragung wird dabei nicht auf exklusive Kanäle beschränkt, sondern alle Daten sämtlicher Nutzer werden nacheinander über denselben Kanal geschickt. Nicht genutzte Kapazitäten einzelner Nutzer stehen allen anderen zur Verfügung. (LFM-NRW: GPRS 2010)

GPRS gehört, wie der weiterentwickelte Standard EDGE[19], zur 2.5-ten Generation von Mobilfunknetzen (2.5G-Netz).

[16] International Mobile Equipment Identity
[17] Personal Identification Number
[18] Subscriber Identity Module
[19] Enhanced Data Rates for GSM Evolution

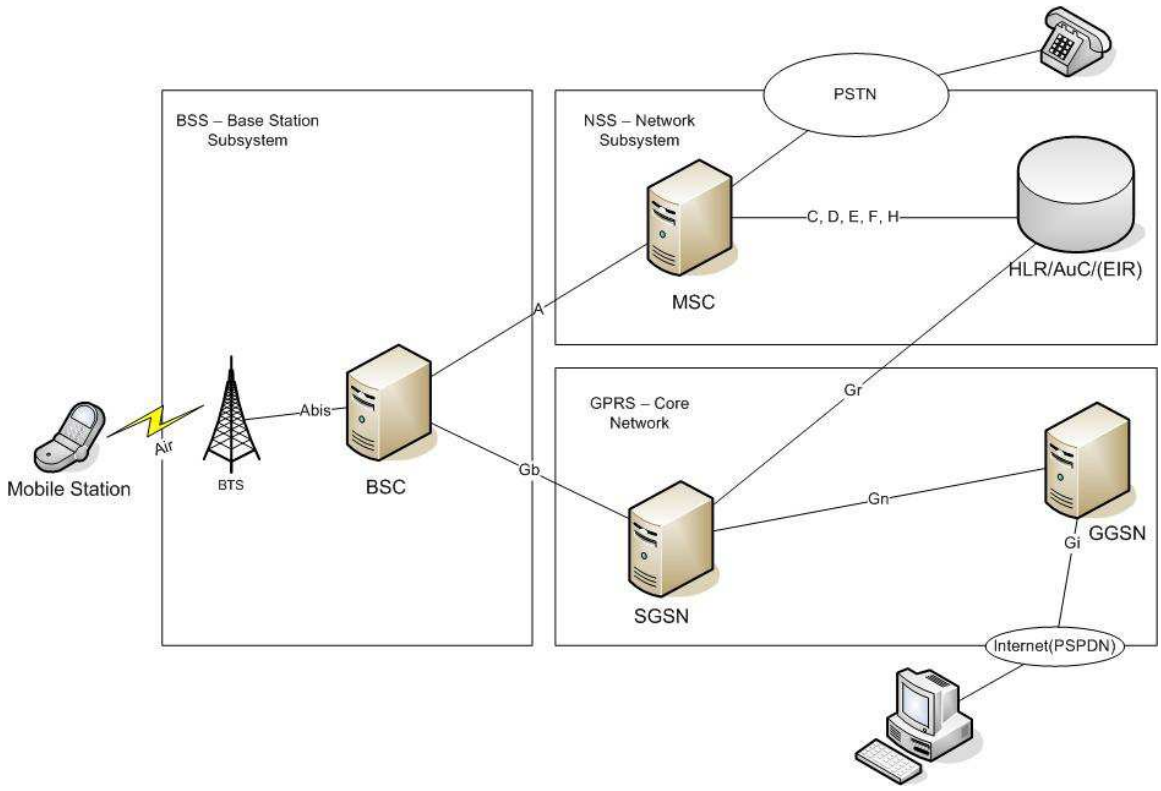

Abbildung 2: 2.5G GPRS

Wie in Abbildung 2 erkennbar, wurde das bestehende GSM-Netz um das „GPRS-Core-Network“ erweitert. Zu diesem Subsystem gehören der SGSN[20] und der GGSN[21]. Der SGSN bildet ein äquivalent zum MSC im Network Subsystem. Während das MSC leitungsorientierte Vermittlungsaufgaben realisiert, verrichtet der SGSN ähnliche Aufgaben für paketorientierte Dienste.

Dazu gehören:

- Aufbau einer Datensession
- Mobilitätsmanagement für alle Teilnehmer seines Zuständigkeitsbereiches
- Bereitstellung von Vergebührungsdaten an Verrechnungsstellen
- Weiterleitung von Datenpaketen über Router
- Weiterführende Aufgaben wie Verschlüsselung, Kompression der Datenpakete, Authentisierung, etc.

Der GGSN dient als Gateway in GPRS- und UMTS-Netzen. Er koordiniert den Datenverkehr zwischen externen paketvermittelten Übertragungsnetzen und Vermittlungsnetzen des

[20]Serving GPRS Support Node
[21]Gateway GPRS Support Node

Mobilfunknetzes(GPRS, UMTS). Eventuell auftretende unterschiedliche Datentechniken oder Datenraten werden mithilfe des GGSN aufeinander abgestimmt. Für das externe PSPDN[22] erscheint das Mobilfunknetz als IP-Netzwerk, welches über den GGSN als Router mit dem PSPDN verbunden ist.

2.1.3 3G Universal Mobile Telecommunication Systems (UMTS)

Die sogenannten 3G-Netze mit UMTS als wichtigsten Vertreter vereinigen die Vorzüge von leitungsvermittelten sprach- und paketvermittelten Datennetzwerken. (LFM-NRW: GPRS 2010)

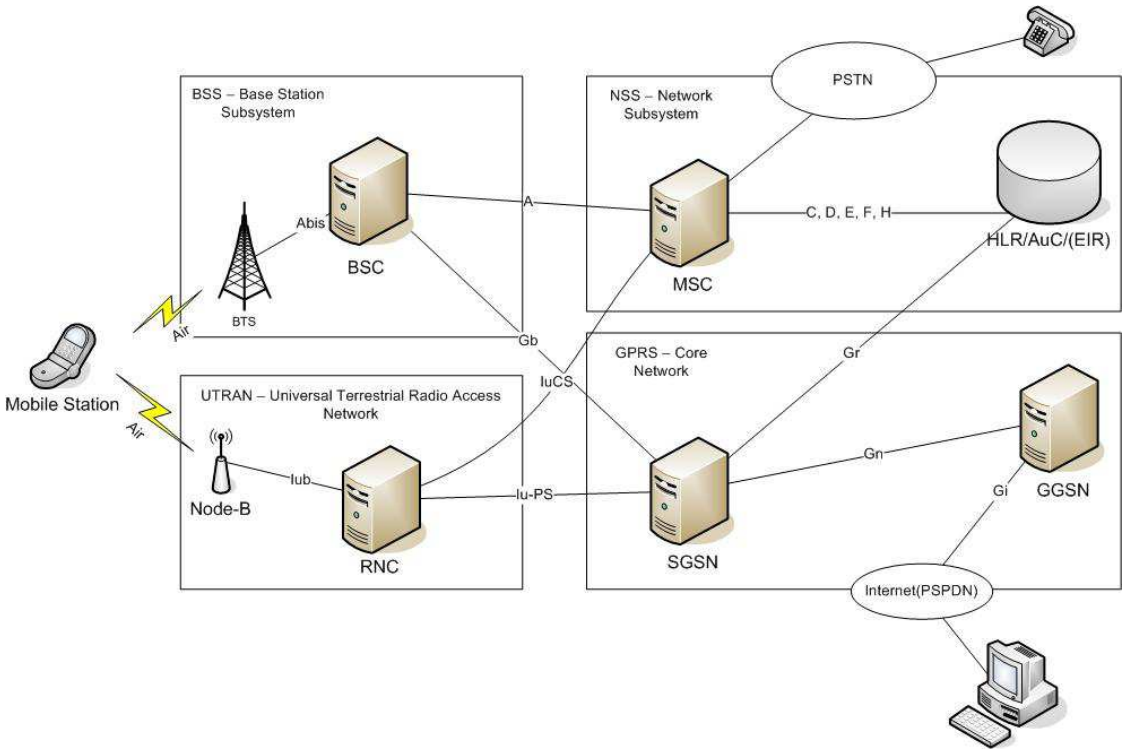

Abbildung 3: 3G UMTS_Rel99

Das 3G-Netz besteht aus den bisher bereits bekannten Subsystemen BSS, NSS und GPRS-Core-Network sowie dem ergänzenden UTRAN[23] (siehe Abbildung 3). Obwohl der Aufbau dem des GERAN's ähnelt, wird eine andere, völlig neue Funktechnik, eingesetzt. Alle funkspezifischen Aufgaben und Prozesse werden im Funknetzwerk und alle dienstspezifischen Prozesse im Kernnetz (Core Network) verarbeitet. Somit ist bei UMTS eine funktional bessere Gliederung des Gesamtnetzes gegeben. Das UTRAN besteht aus

[22] Packet Switched Public Data Net
[23] Universal Terrestrial Radio Access Network

Node-B und RNC[24]. Die Node-B ist das Äquivalent zur BTS und der RNC das äquivalent zum BSC aus dem GSM-Standard. Mehrere Node-B sind an einer RNC angeschlossen und kommunizieren über die auf ATM[25]-basierenden Schnittstelle „Iub“. Anders als im GSM-Netz kommunizieren im UTRAN die RNC’s auch untereinander über die „Iur“ Schnittstelle. Somit wird das „Handover“ innerhalb des UTRAN möglich. Teilnehmer einer Funkzelle können mittels „Roaming“ in einen anderen Funkzellenbereich wechseln. (Riemer, R. 2010)

Die Leistung von UMTS-Netzen wurde durch verschiedenste Ausbaustufen verbessert. So entwickelte man HSDPA[26], welches eine theoretische Downloadrate von 14,4 Mbit/s ermöglicht und analog dazu HSUPA[27], welches eine maximale Upload Geschwindigkeit von 5,67 Mbit/s ermöglicht. Die höchste Ausbaustufe HSPA+[28] bietet unter Idealbedingungen eine Datenübertragungsrate von 28Mbit/s(DL) und 11,5Mbit/s(UL). (LFM-NRW: GPRS 2010)

2.1.4 4G Next Generation Mobile Network (NGMN) (Wikipedia: NGMN 2010)

Zur Entwicklung der nächsten Mobilfunkgeneration wurde ein Projekt von mehreren Mobilfunkfirmen und Ausrüstern ins Leben gerufen.

„NGMN setzt auf der UMTS-Infrastruktur auf und soll so eine rasche und kostengünstige Erweiterung zum bestehenden 3G-Netz darstellen. Realisiert werden soll eine Geschwindigkeit von 100 Mbit/s. Außerdem soll die Möglichkeit eines „always on“ Modus geschaffen werden, mit dem Endgeräte permanent mit dem Internet verbunden bleiben können. Erreicht werden soll dies durch eine bessere Ausnutzung des zur Verfügung stehenden Frequenzspektrums. Eine einfachere Netzwerkarchitektur wird so ebenfalls ermöglicht und führt zu Latenzzeiten unterhalb von 10 ms.“ (Wikipedia: NGMN 2010)

[24] Radio Network Controller
[25] Asynchronous Transfer Mode
[26] High Speed Downlink Packet Access
[27] High Speed Uplink Packet Access
[28] High Speed Packet Access

Abschließend werden in der folgenden Tabelle 1 alle bisherigen Mobilfunkgenerationen mit ausgewählten Vertretern in einem kurzen Überblick dargestellt.

Generation	Mobilnetz	Übertragung	Vermittlung	Datenübertragung
1G	AMPS	Analog	Leitungsvermittelt	Keine
2G	GSM HSCSD	Digital Digital	Leitungsvermittelt Leitungsvermittelt	9,6kbit/s;14,4kbit/s 76/115 kbit/s
2.5G	GPRS EDGE	Digital Digital	Paketvermittelt Paketvermittelt	171 kbit/s 473 kbit/s
3G	UMTS	Digital	Paketvermittelt	2 Mbit/s
3.5G	HSDPA HSUPA	Digital Digital	Paketvermittelt Paketvermittelt	14 Mbit/s (DL) 5 Mbit/s (UL)
3.9G	LTE	Digital	Paketvermittelt	
4G	UTRAN LTE Advanced UWB EVDO	Digital	Paketvermittelt	>> 100 Mbit/s

Tabelle 1: Übersicht Entwicklung Mobilfunk (ITWissen_Lexikon: NGMN 2010)

Die folgenden Erläuterungen zum Thema Verbindungsaufbau und Intelligentes Netz sollen dem Leser eine Vorstellung von Mobilfunknetzen, dessen Funktionsweise und der immer weiter zunehmenden Komplexität vermitteln.

2.2 Verbindungsaufbau am Modell eines Mobile Mobile Calls

In der folgenden Abbildung 4 wird modelhaft ein Verbindungsaufbau zwischen zwei Mobilteilnehmern dargestellt und anschließend näher erläutert. Es wird davon ausgegangen, dass Mobilteilnehmer „A“ (im Folgenden „TLN_A“ genannt) Mobilteilnehmer „B“ (im Folgenden „TLN_B“) anruft.

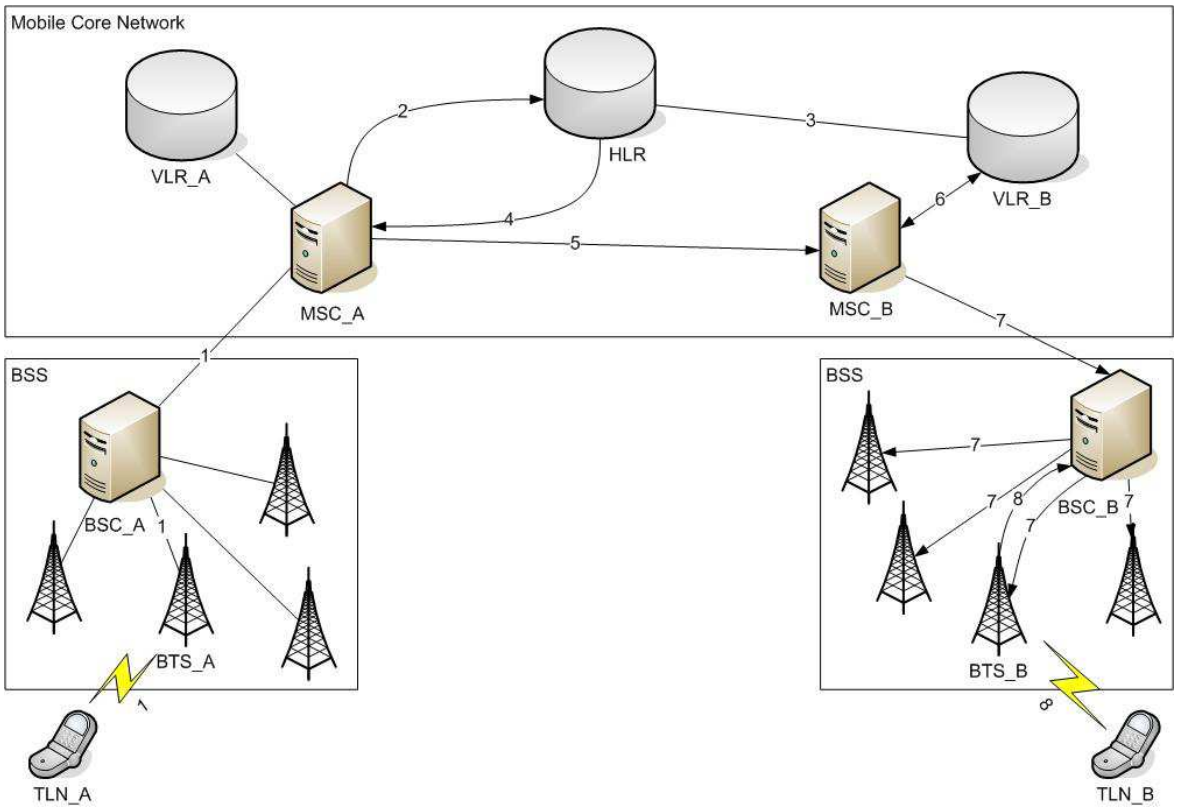

Abbildung 4: Mobile To Mobile Call im PLMN

1. A-Tln ruft B-Tln --> Verbindung zu A-MSC
2. A-MSC sendet Anfrage an HLR
3. HLR holt sich von B-VLR die Mobile Station Roaming Number
4. HLR übergibt MSRN an A-MSC
5. A-MSC verbindet mithilfe der MSRN mit B-MSC
6. B-MSC überprüft Aufenthaltsort des B-Tln über Abfrage des B-VLR
7. B-MSC sendet über BSC ein „Paging Signal“
8. B-Tln antwortet und meldet so seine Funkzelle --> Funkkanal kann geschalten werden

TLN_A und TLN_B befinden sich in unterschiedlichen Funkzellen und sind somit an unterschiedlichen BTS's angemeldet. TLN_A wählt die IMSI des TLN_B auf seinem Mobilteil. Die Signalisierung wird über die BTS_A an die BSC_A weitergeleitet und gelangt von dort über das A-Interface an die MSC_A. Die MSC des TLN_A verbindet sich nun zum HLR. Das HLR holt sich die MSRN (Mobile Station Roaming Number) von der VLR_B und übergibt diese an die MSC_A. Über die MSRN verbinden sich nun MSC_A und MSC_B. Anschließend überprüft die MSC_B, ob sich der TLN_B noch im Einzugsgebiet der VLR_B befindet (Identitätsprüfung + Anforderung des gegenwärtigen Aufenthaltsort). Anschließend sendet die MSC_B ein Paging-Signal an alle BSC's die an der MSC_B angeschlossen sind und damit an alle angeschlossenen BTS's. Empfängt nun der TLN_B dieses Paging-Signal über eine der BSS's, schickt er eine Antwort. Ist der TLN_B gerade in einem Gespräch, wird dem TLN_A ein Besetztzeichen signalisiert. Ist TLN_B in keinem Gespräch, bekommt TLN_A das Freizeichen signalisiert und am TLN_B erfolgt die Ruftonsignalisierung. Bestätigt der TLN_B indem er den „Hörer abnimmt", wird ein Sprachkanal über die beiden MSC's geschaltet und die Verbindung von TLN_A zu TLN_B besteht.

2.3 Dienste im GSM-Netz

Im GSM-Netz sind genau wie im Festnetz die Dienste (Services) in die bekannten 3 Kategorien eingeteilt:

- Bearer Services
    - Ermöglichen Transport von Nachrichten über Netze
    - 3 GrundlegendeTypen:
        - Leitungs- bzw. kanalvermittelt (circuit switched)
        - Paketvermittelt (packet switched)
        - Festverbindungen (leased lines)
- Teleservices
    - Z.B.: Telefonie, E-Mail, WWW, Voice over IP, FTP, etc.
- Supplementary Services
    - Ergänzen Teleservices
    - Zusatzdienste wie z.B.: Rückfrage, Makeln, Konferenz, Rufweiterleitung, etc.

2.4 IN – Intelligente Netze (ITWissen_Lexikon : IN 2010); (Peter 2010); (UMTSLink: Intelligent Network 2010)

Im zunehmenden Maße werden Dienste benötigt, die speziell auf Kundenbedürfnisse zugeschnitten sind. Um diesem Anspruch gerecht zu werden, wurde IN entwickelt. In IN-Netzen wird eine offene Plattform für die Entwicklung, Bereitstellung und Management von Diensten geschaffen.

Folgende Punkte sind dabei entscheidend:

- Neue Dienste schnell und unabhängig vom Trägernetzwerk implementierbar
- Einheitliche Protokolle und Schnittstellen
- Dienstekonfigurationen in zentraler Datenbank
- Bereitstellung TK-Dienste entkoppelt vom physikalischen Vermittlungsnetz

Ein Intelligentes Netz ist keine spezielle physikalische Netzplattform, sondern eine zusätzliche diensteorientierte Architektur, die auf bereits existierenden NW-Infrastrukturen aufsetzt und diese mit zusätzlichen Leistungsmerkmalen (IN-Dienste) ausstattet. Aus Anwendersicht handelt es sich um eine Vielzahl von Leistungs- und Komfortmerkmalen, die eine Optimierung der Telekommunikationsabläufe ermöglichen.

Beispiele solcher IN-Dienste sind u.a.:

- Notrufnummern
    - 110/112 für Polizei und Rettungskräfte
- Televoting
    - Abstimmungen per Telefon, bekannt aus dem TV
- Telephone number portability
    - Rufnummernmitnahme beim Wechsel des Mobilfunkanbieters
- Toll free calls
    - Dienste die kostenfrei zur Verfügung gestellt werden z.B.: Servicerufnummer zur Sperrung der VISA-CARD: 0800/8118440

IN-Netze sind unabhängig von den angebotenen Diensten und vorhandenen Netzen. Für den Endnutzer bestehen also keine spürbaren Einschränkungen bzw. Unterschiede zwischen Diensten aus dem Festnetz und Diensten aus dem Mobilfunknetz.

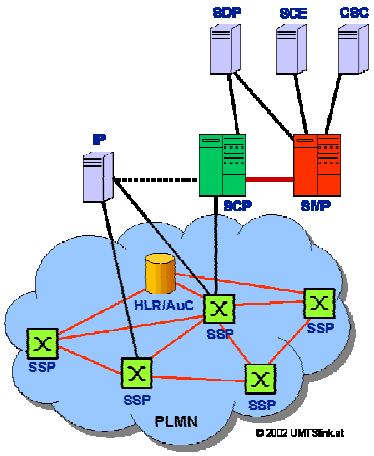

Abbildung 5: Intelligent Network (UMTSLink: Intelligent Network 2010)

Abbildung 5 zeigt die Basisarchitektur eines Netzwerkes mit IN-Erweiterung. IP steht hierbei für „Intelligent Peripheral“ und beschreibt zusammenfassend Erweiterungen, die für die Nutzung von IN-Diensten erforderlich sind. Neu entwickelte und bereits existente Dienste werden im SCP[29] zentral eingelagert und sind somit im kompletten Netzwerk verfügbar und steuerbar. Der SSP[30] beschreibt den Zugangspunkt für Dienstnutzer. Im Mobilen IN ist die Funktion des SSP in der MSC integriert. IN-Rufe (Rufe mit speziellem Dienstaufruf) werden hier erkannt und die Rufüberwachung übernommen.

Der STP[31] ist ein paketorientierter Switch im Signalisierungsnetzwerk, welcher Kontrollsignale zwischen verschiedenen Elementen im Netzwerk steuert. So z.B. zwischen MSC/SSP und SCP.

Alle Dienste eines Intelligenten Netzes werden im SCP gesammelt und bereitgestellt.

IN basiert auf dem „Frage – Antwort“ Konzept. Ein Beispiel könnte so aussehen: Eine MSC, die mit IN-Logik ausgestattet ist, stellt eine Dienstanfrage an einen SCP. Der SCP bearbeitet die Anfrage und schickt eine Antwort zurück an die MSC, welche daraufhin ihren Call Prozess wie geplant fortsetzt. (MobileIN.com 2010)

Die klassische Architektur der IN-Netze ist in den Standards ITU[32]-T Q.1200ff. definiert. Varianten für den Mobilfunk finden sich in den Spezifikationen des 3GPP[33].

[29] Service Control Point
[30] Service Switching Point
[31] Signalling Transfer Point
[32] International Telecommunication Union
[33] 3rd Generation Partnership Project

2.5 Testarten (Uni Zürich: Software engineering 2010)

„Ein Softwaretest ist ein Test während der Softwareentwicklung, um die Funktionalität einer Software an den Anforderungen und ihre Qualität zu messen, und Softwarefehler zu ermitteln.“ (Wikipedia, Softwaretest 2010)

Um ein System zu testen werden verschiedene Testarten verwendet. Diese werden im Folgenden aufgelistet und stichpunktartig erläutert:

1. Protokolltest
   - Test von Schnittstellen und Protokollen.
2. Regressionstest
   - Überprüfung bereits getesteter Testfälle
   - Vergleichen und eventuelle Unterschiede auswerten
3. Funktionstest
   - Tests spezieller Funktionen z.B.: Durchführung IN-Rufszenarien, Administration der Dienstparameter, Service Management, Statistiken, etc.
4. Robustheits- und Negativtest
   - Hier wird die Reaktion des Systems im Fehlerfall getestet. Bei Eintreten von Fehlern soll das System in einer definierten Art und Weise reagieren.
5. Dauertest und Lasttest
   - Das System wird hierbei über einen längeren Zeitraum und unter möglichst hoher Auslastung getestet.
6. Betreibertest
   - Unter Betreibertests versteht man Tests von administrativen Aufgaben wie: Back up und Restore, geregelter Restart der Anlage, Fehlerbehandlung und Alarmierung, Softwarewechsel im laufenden Betrieb und Installationstest.
7. Stresstest
   - Hier wird der Ausfall von Komponenten simuliert, z. B.: Hardwaredefekt

2.6 Tests bei NSN (Alekseev 2010), (Uni Zürich: Software engineering 2010)

Wie bereits in der Einleitung dieser Bachelorarbeit erwähnt, ist es eine der Hauptaufgaben der Abteilung verschiedenste Tests im Bereich der Mobilfunktechnik durchzuführen. Zum allgemeinen Verständnis soll im Folgenden der allgemeine Testvorgang kurz näher erläutert werden.

Auf der Grundlage von Verträgen mit Netzbetreibern oder Dienstanbietern werden zu allererst Testprojekte definiert. Währenddessen wird den Entwicklungsabteilungen ein kundenspezifischer Konfigurationsauftrag erteilt. Auf Grundlage dieses Auftrages wird eine Teststrategie entwickelt und daraus die durchzuführenden Testfälle definiert. Nachdem das System und die kundenspezifische Testanlagenkonfiguration bereitgestellt wurden, wird mit der Testdurchführung begonnen.

Die Tests werden solange durchgeführt bis alle definierten Zielkriterien erfüllt wurden. Anschließend muss die Qualitätssicherung noch ihr „OK“ geben bevor die Auslieferung an den Kunden erfolgt.

2.6.1 Testablauf

Um Testabläufe erfolgreich durchzuführen müssen die Ziele des Testens klar sein. Aus diesem Grunde werden geforderte Funktionen (Ziele) an das Testsystem in Anforderungsspezifikationen definiert.

Testfälle werden grundsätzlich in 3 Teile untergliedert:

- Testvorbereitung/ -planung
- Testausführung
- Testauswertung

2.6.2 Testvorbereitung/ -planung

In dieser Phase der Tests werden alle grundlegenden Fragen geklärt und alle Voraussetzungen geschaffen, um die definierten Ziele des Testfalles erreichen zu können. Dazu gehören:

- Bereitstellung benötigter Ressourcen
- Werkzeuge
- Methoden
- Richtlinien für Tests
- Bestimmung verantwortlicher Personen
- Beschreibung von Testfällen

2.6.3 Testausführung

Wurden alle Voraussetzungen geschaffen, werden nun die definierten Testfälle ausgeführt und die Ergebnisse dokumentiert.

2.6.4 Testauswertung

Nach der Durchführung von Testfällen erfolgt die Auswertung. Dabei wird vor allem ein „Soll-Ist-Vergleich“ durchgeführt um eventuelle Abweichungen der Ergebnisse von den vorher definierten Zielkriterien zu analysieren. Weichen die Ergebnisse von den Zielen ab, dann werden die Testfälle solange wiederholt, bis sie innerhalb der definierten Zielkriterien funktionieren. Sobald alle Zielkriterien erfüllt sind, kann der Testfall abgeschlossen werden.

3 Analyse vergleichbarer Entwicklungen

In diesem Kapitel werden ausgewählte, bereits entwickelte Softwarelösungen vorgestellt und analysiert und im Anschluss daraus Erkenntnisse für die Entwicklung der Softwarelösung des Autors gezogen.

3.1 NetActSystem 6.0 Monitor (Nokia Siemens Networks 2010)

Dieses von Nokia Siemens Networks entwickelte Monitorsystem, kurz NAS genannt, ermöglicht es, Alarme verschiedenster Netzwerkelemente und Typen zu verwalten und dabei die Hauptursachen zu analysieren. Außerdem bietet der NAS 6.0 Monitor die Möglichkeit, Fehlerfälle, die zu Unterbrechungen in Netzwerk Services führen, zu bearbeiten und zu lösen. Die Qualität der Netzwerk Services wird somit für den Nutzer verbessert.

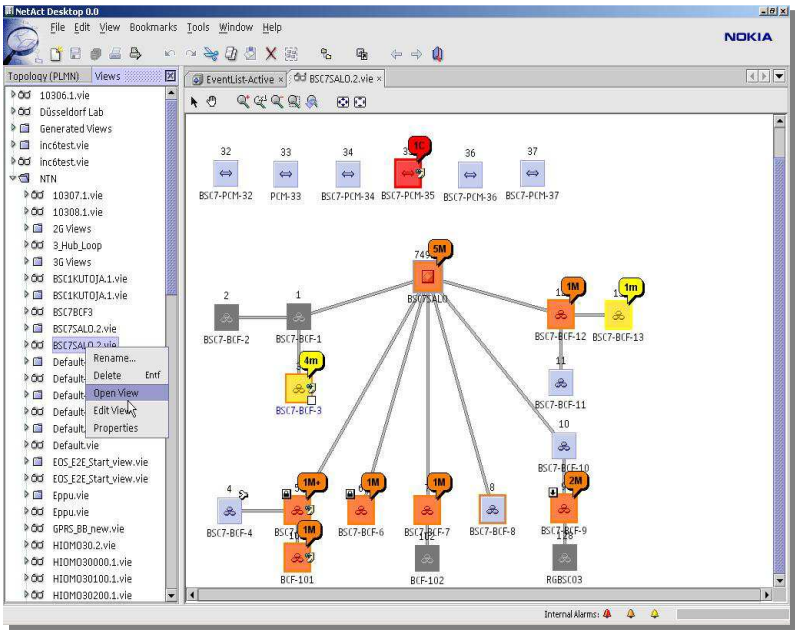

Abbildung 6: NAS 6.0 Monitor (Nokia Siemens Networks 2010)

Folgend werden weitere Funktionen des NAS 6.0 Monitors benannt:

- Weiterleitung von Alarminformationen zur Weiterverarbeitung an externe Tools
- Untersuchen und Verwalten von Alarmsituationen mit flexiblen Tools
- Filtern und Zusammenhänge erkennen um das Wesentliche eines Fehlerfalles zu bestimmen
- Speichern von Alarmevents in einer Datenbank

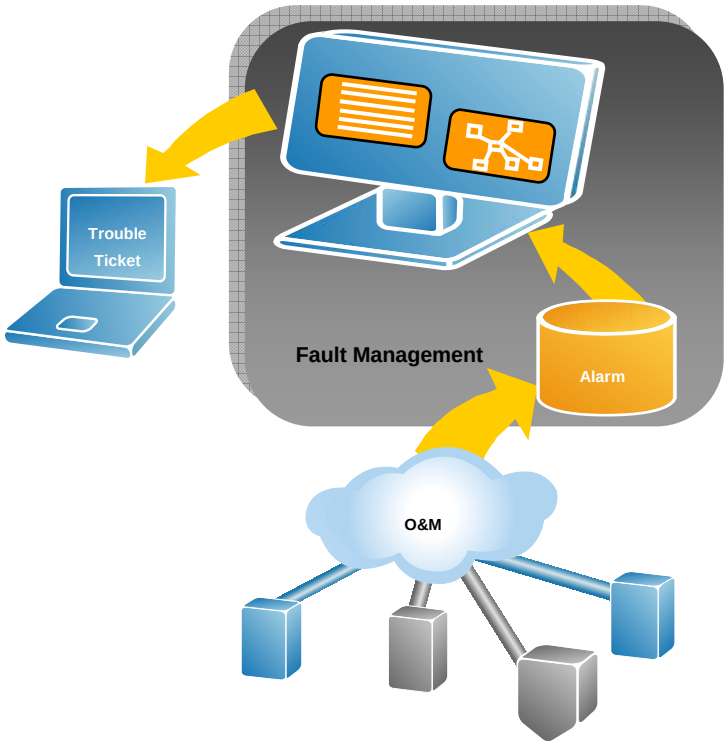

Abbildung 7: Ablauf Sammeln von Alarm Events (Nokia Siemens Networks 2010)

3.2 Nagios (Wikipedia, Nagios 2010)

Nagios wird für die Überwachung von verschiedenen Netzwerkdiensten verwendet. So z. B.: SMTP[34], POP3[35], HTTP[36] oder NNTP[37] auf Servern. Wird ein Problem erkannt, dann erfolgt eine Benachrichtigung an den Nutzer, je nach Wunsch per E-Mail, IM[38] oder SMS[39]. Der Netzwerkstatus und auch die entsprechenden LogFiles lassen sich auf Wunsch im Web Browser betrachten.

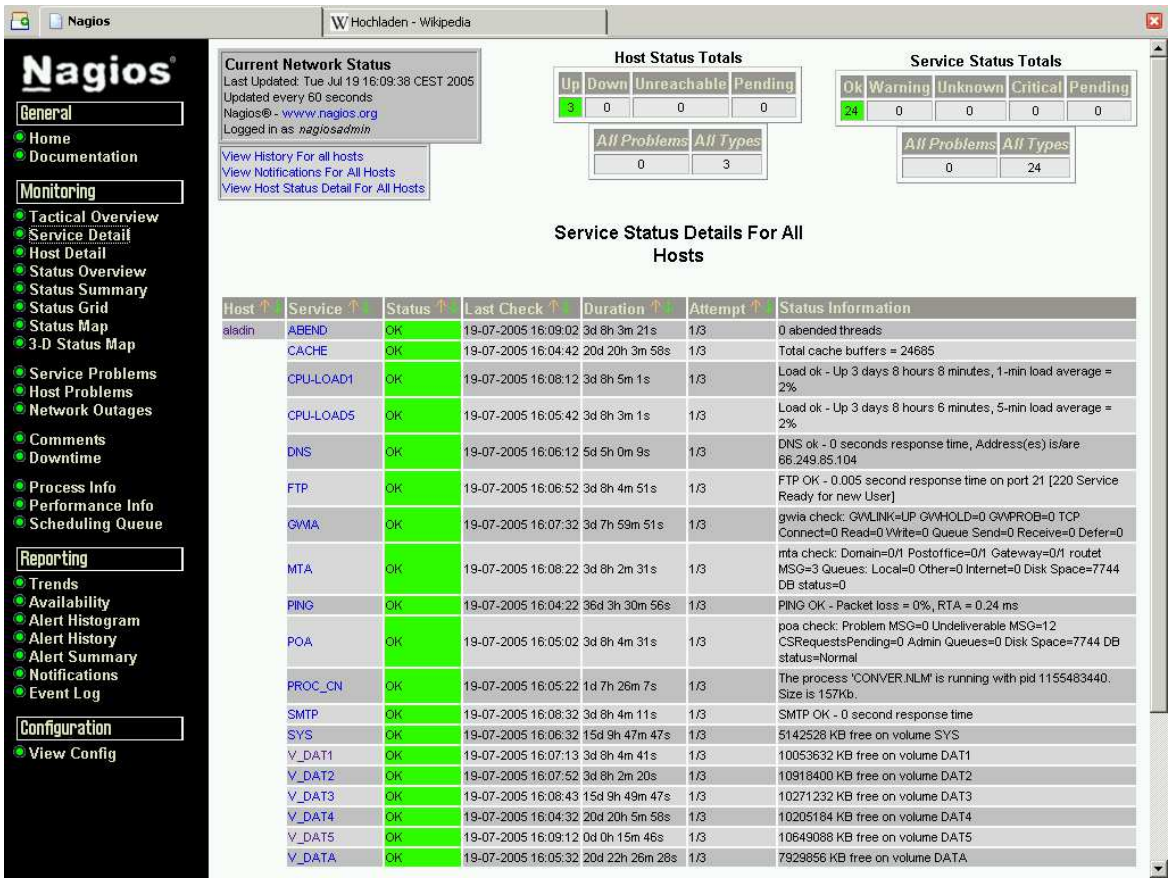

Abbildung 8: Nagios Visualisierung (Wikipedia 2010)

[34] Simple Mail Transfer Protocol
[35] Post Office Protocol
[36] Hypertext Transfer Protocol
[37] Networks News Transfer Protocol
[38] Instant Messaging
[39] Short Message Service

3.3 Big Brother Professional Edition (Quest Software: BigBrother 2010)

Big Brother wird für die Überwachung von Hosts und TCP[40]-Diensten in TCP/IP[41]-Netzen eingesetzt. Dabei werden regelmäßig der Systemzustand und die Netzwerkverbindungen von PC's in einem Netzwerk überprüft. Alle Überwachungsergebnisse werden in Tabellenform zusammengefasst dargestellt und können in einem Browser aufgerufen werden. Treten an den überwachten Rechnern Fehler auf, können nähere Informationen zu den Fehlern, per Mausklick auf dem Rechner, gesammelt werden. Neben den üblichen Netzwerküberwachungen ist Big Brother ebenfalls in der Lage, die CPU- oder Festplattenauslastung zu überwachen. Big Brother arbeitet als Server-Client-System. Während der Server unter Linux laufen muss, ist bei den Clients Linux und Windows NT möglich. Es ist außerdem möglich, Grenzwerte zu definieren, die bei Überschreitung eine Alarmierung auslösen.

[40] Transmission Control Protocol
[41] Internet Protocol

3.4 MRTG – Multi Router Traffic Grapher (Heise MRTG 2010)

In erster Linie ist MRTG, wie der Name schon sagt, für die Überwachung von Routerfunktionen gedacht. Über SNMP[42] wird die Auslastung von Servern und Routern überwacht. Anschließend wird der Netzwerkverkehr auf HTML-Seiten in Form von Grafiken präsentiert.

Mit MRTG können aber auch andere Netzwerkkomponenten überwacht und visualisiert werden. Von Wetterdaten bis Snackmaschinen ist dabei alles möglich. Mithilfe von speziellen Skripten ist auch die Überwachung von Geräten ohne SNMP Unterstützung möglich. So z.B. die CPU-Auslastung auf Windowssystemen.

MRTG ist kostenlos und arbeitet unter vielen der üblichen Betriebssysteme wie z. B.:
Windows NT, 2000, XP, 2003, Vista, 7, Linux, Mac OS X, Mac OS X/Intel

Beispiel:

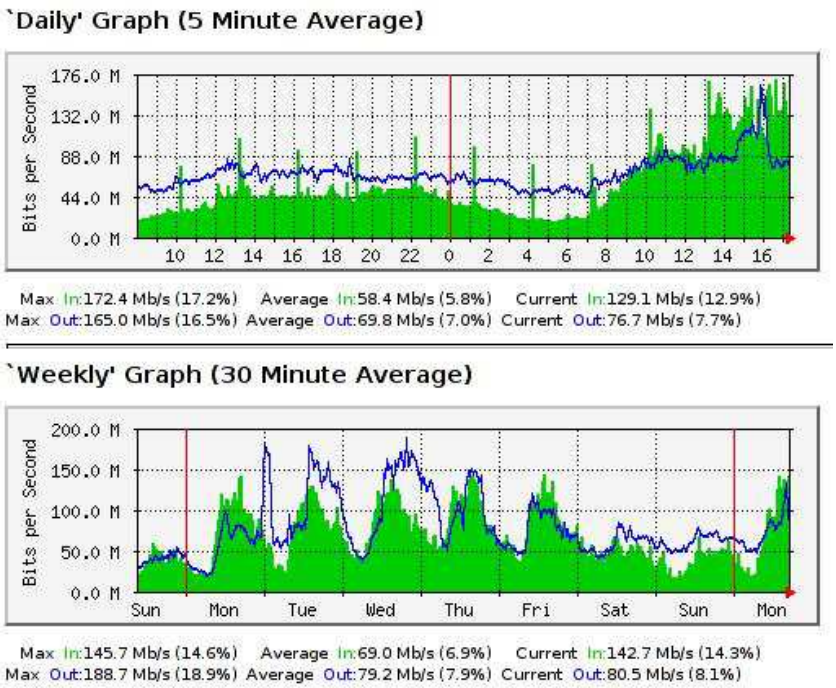

Abbildung 9: Beispiel MRTG (Heise MRTG_Bild 2010)

[42]Simple Network Management Protocol

3.5 Hobbit („Xymon“) (Heise: Xymon 2010)

„Xymon überwacht Rechner im Netzwerk, die darauf bereitstehenden Dienste und die Netzwerkverbindung zwischen ihnen“. (Heise: Xymon 2010)

Die aus der Überwachung heraus erstellten Berichte werden als Web-Seiten visualisiert und regelmäßig mit den aktuellen Statusinformationen aktualisiert.

Xymon ist eine Weiterentwicklung aus einem Plugin für Big Brother. Da Xymon kostenlos zur Verfügung steht, ist es eine gute Alternative zum kommerziellen Big Brother. Meist wird Xymon in Rechenzentren eingesetzt um eine größere Zahl von Rechnern zu überwachen. Im Allgemeinen ist Xymon aber auch in der Lage kleine Netzwerke zu überwachen.

Ebenfalls überwacht werden können die Festplattennutzung, Logfiles und Prozesse. Dies erfolgt über Agents die auf dem Xymon Server installiert sind. Alle Überwachungsergebnisse werden von einem zentralen Server gesammelt und genutzt um Web-Seiten zu erstellen, die den Status des Netzwerkes anzeigen. Xymon speichert ebenfalls die Historie von jedem überwachten Item. So kann man Verfügbarkeitsreports verfassen und Vorfälle kontrollieren, die irgendwann einmal aufgetreten sind. Es werden ebenfalls Daten zur Erstellung von Trendanalysen, dargestellt als Graph, gespeichert. So kann man z. B. leicht die Antwortzeit von geschäftskritischen Web Application aufzeichnen. Alarmnachrichten können per Mail, SMS oder Pager versendet werden. Dies ermöglicht die schnelle Reaktion auf Probleme, ohne ständig auf den Service aufzupassen.

Abbildung 10: Beispiel Xymon (Xymon 2010)

3.6 Erkenntnisse

Alle vorgestellten Softwarelösungen verwenden Web-Seiten als visuelles Medium. Zustandsmeldungen und andere Statistiken können so vom Nutzer erfasst werden, ohne ein entsprechendes Programm auf dem eigenen Rechner starten zu müssen. Dies spart Ressourcen und ist somit auch für die vom Autor entwickelte Software ein erster Ansatz. Die meisten Überwachungstools sind darauf ausgelegt, auf separaten Servern zu arbeiten. Über IP-Verbindungen werden Daten der entsprechend zu überwachenden Netzwerkelemente gesammelt, analysiert und anschließend mithilfe von Web-Seiten visualisiert. In der Mobilfunktechnik werden einige Netzelemente nicht direkt über IP-Schnittstellen verwaltet. Es existieren verschiedenste Tools, die für die Konfiguration und Wartung verwendet werden. So wird z. B. das MSC über den sogenannten Switch Commander konfiguriert. Anders als bei den vorgestellten Tools im vorangegangenen Kapitel ist es für die Realisierung der Überwachung von Netzelementen (im Folgenden auch kurz als „NE“ bezeichnet) in der Mobilfunktechnik also notwendig, zu Beginn eine geeignete universelle Schnittstelle zu definieren.

Jedes NE hat unterschiedlichste Eigenschaften, die entscheidend sind für die Aussage, ob ein NE in Ordnung ist oder nicht. Für jedes NE muss also separat entschieden werden, welche Eigenschaften und Funktionen verfügbar sein müssen, um wesentliche Fehler oder Funktionsstörungen auszuschließen.

Des Weiteren müssen nun die Zustandsmeldungen der Netzelemente ausgewertet und verarbeitet werden. Wichtig ist dabei, die projektspezifische Sicht auf die Netzelemente zu beachten. Nicht jedes NE wird in jedem Projekt verwendet. Es muss also eine Zuordnung von NE zu aktuellen Projekten erfolgen.

Neben den Zustandsabfragen einzelner Netzelemente soll außerdem die Möglichkeit gegeben werden, die Funktion eines ganzen Projektes zu testen. Betrachtet man ein Projekt, ähnlich wie ein einzelnes Netzelement, als eine Sammlung komplexer Tests, so ermöglicht diese Sichtweise ein ähnliches Testvorgehen wie bei den Einzelelementen. Bestimmte Eigenschaften des Projekts müssen definiert und getestet werden, um eine Aussage zum Zustand des Gesamtprojektes zu geben.

Zur Visualisierung der einzelnen Projekte gehört auf jeden Fall ein Übersichtsbild mit der genauen Anordnung aller Netzelemente eines Projektes. In folgender Abbildung 11 wird dargestellt, wie eine solche Projektübersicht aussehen könnte.

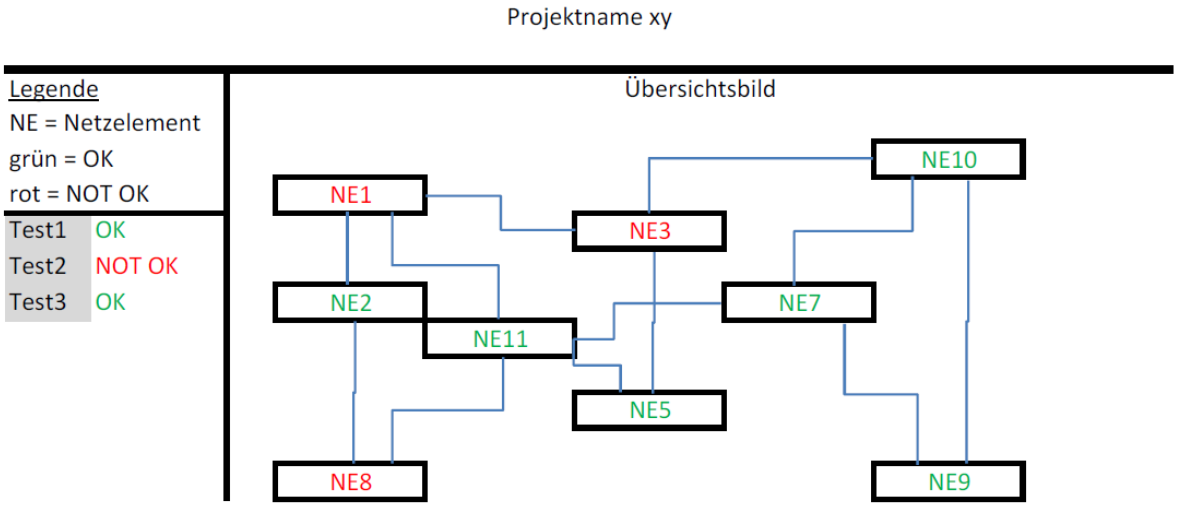

Abbildung 11: Entwurf einer Projektübersicht

Der Grundgedanke war hierbei, durch farbliche Änderung der Hintergründe eines jeden einzelnen Netzelementes den jeweils aktuellen Zustand anzuzeigen. So würde man eine gute Übersicht über das Projekt mit allen aktuellen Zuständen der verwendeten NE erhalten. Ergebnisse der Tests des Gesamtprojektes in einer Tabelle, geordnet nach Komplexität des ausgeführten Testfalls, würden die Projektdarstellung vervollständigen.

4 Entwurf der Software

In diesem Kapitel erfolgt der Softwareentwurf. Dieser basiert auf den gewonnenen Erkenntnissen aus Kapitel 3 sowie den Anforderungen der Abteilung an die Eigenschaften der Software.

4.1 Grundgedanken zur Softwareentwicklung

In Abbildung 12 wird das vom Autor entwickelte Lösungskonzept zur Entwicklung der Software dargestellt und nachfolgend näher erläutert.

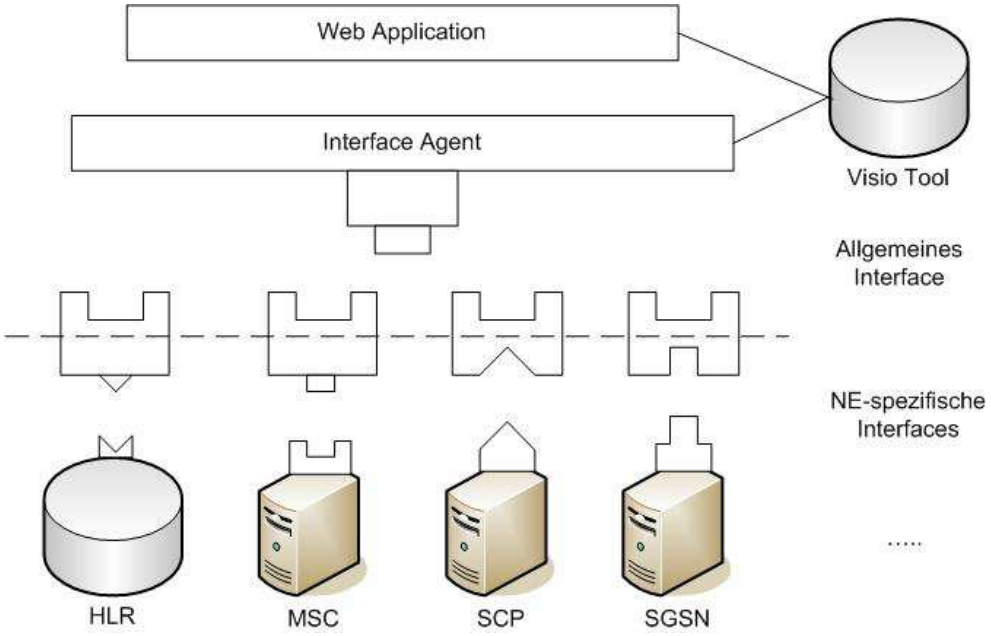

Abbildung 12: Vorüberlegung Allgemeine Schnittstelle

Diese Lösung basiert auf einer Web-Applikation, wodurch die Entwicklung und die Installation einer Spezialsoftware entfallen. Ziel ist es, dass im Webbrowser ein Testszenario ausgewählt werden kann und dieses dann als Abbildung dargestellt wird. Die Testszenarien kommen aus einer Datenbasis, die mit Microsoft Visio bearbeitet wurden. Für jedes Szenario existieren eine Abbildung im JPEG-Format und ein XML-Datensatz. Anhand des XML-Datensatzes erfährt die Web-Applikation, welche Netzelemente vorhanden sein müssen. Nun ist es Aufgabe der Web-Applikation, die Zustandsdaten der Netzelemente zu ermitteln und anzuzeigen.

Wie aus der Abbildung 12 hervorgeht, ist die Netzelementewelt nicht homogen. Der Autor hatte deshalb die Aufgabe, eine allgemeingültige Schnittstelle zwischen Applikation und Netzelementewelt zu definieren.

Ein grundlegendes, zu klärendes Problem für die Zustandsanzeige komplexer heterogener Systeme war, die verschiedensten Netzelemente mit ihren unterschiedlichsten Zugriffs- und Auswertungsmethoden auf eine einheitliche Art und Weise abzufragen und so deren Zustände in Erfahrung zu bringen.

Das Ziel war also die Definition einer einheitlichen Schnittstelle für die Sammlung aller aktuellen Zustände der NE. Diese Daten sollen dann unabhängig vom Netzelement und den durchgeführten Tests zur Verfügung gestellt werden.

Als unkomplizierteste Variante der Datensammlung entwickelte sich die Variante, für jedes zu überwachende Netzelement eine Textdatei erzeugen zu lassen, in der dann definierte Angaben über den Zustand des NE und andere relevante Informationen hinterlegt werden können. Das Dateiformat und das Format, indem Informationen innerhalb der Dateien hinterlegt werden, werden fest definiert um eine Auswertung einfach und unkompliziert durchführen zu können. Inhalt der NE-Dateien muss auf jeden Fall der aktuelle Zustand des NE sein.

Bei jedem Netzelement gelten dabei natürlich andere Kriterien zur Bestimmung des aktuellen Zustandes. Diese Kriterien zu jedem einzelnen NE zu bestimmen und zu definieren, würde den Rahmen dieser Arbeit sprengen, deshalb übernahmen dies die jeweiligen Betreuer der Netzelemente in der Arbeitsgruppe. Ihre Aufgabe war es, zu jedem der zu überwachenden NE die entsprechende Datei nach gegebenen Richtlinien anzulegen und in einen definierten Ordner im Netzwerk abzulegen. Dabei ist es Ihnen überlassen, welche Tests sie am Netzelement durchführen um den Zustand zu ermitteln. Wichtig ist dabei die Einhaltung der Richtlinien zum Erstellen der Textdateien.

Mit dieser Herangehensweise ist es möglich, im Laufe der Verwendung der zu entwickelnden Anwendung, weitere NE-Dateien hinzuzufügen und zu entfernen, ohne die folgenden Routinen für die Visualisierung oder der Web Applikation (Abbildung 12) neu definieren zu müssen.

Benutzt man nun ein strukturiertes Filesystem zur Ablage der Dateien, entwickelt sich eine Alternative zu einem komplexen Datenbanksystem. In einem Ordner werden sämtliche Netzelemente, deren Zustand ermittelt wurde, abgelegt. Ein weiterer Ordner enthält die aktuellen Projekte und eine Liste der jeweils verwendeten NE. Die gesammelten Daten der aktuellen Projekte und der NE werden so für die Auswertung der Zustände und der Realisierung der grafischen Übersicht bereitgestellt. In Abbildung 12 beschreibt der „Interface Agent“ diesen Teil des Lösungsansatzes.

Es schließt sich nun das Problem der Visualisierung der gesammelten Daten an. Wie bereits in der Vorüberlegung erwähnt, ist dabei ein Übersichtsbild über das gesamte Projekt, die Zustände der einzelnen NE sowie eine Aussage zur Funktion des Projektes im Ganzen zu realisieren.

In der Arbeitsgruppe wurden bereits in der Vergangenheit zu jedem Projekt Übersichtsbilder mit Microsoft Visio erstellt. Es bot sich daher an, diese gleich als Quelle zu nutzen und so die Visualisierung der Projekte und ihrer Netzelemente zu realisieren. Die aktuellen Zustände der einzelnen Netzelemente im Projekt sollten dann durch entsprechende farbliche Änderung im Übersichtsbild hervorgehoben und in den jeweiligen Projektordnern abgelegt werden. Aufgabe der Abbildung 12 als „Visio-Tool“ bezeichneten Abschnitts ist also das Aufgreifen der vorhandenen Visio Übersichtsbilder, die Visualisierung der Zustände der jeweils im Projekt verwendeten Netzelemente und die korrekte Ablage der Bilder im entsprechenden Projektordner.

Nachdem nun alle Informationen über die aktuellen Projekte und die Zustände der jeweiligen NE gesammelt wurden, müssen diese Ergebnisse für alle Betreuer zugänglich gemacht werden. Dazu bietet sich die Präsentation eines jeden Projektes als Web-Page an. Natürlich will nicht jeder Betreuer alle Projekte sehen. Die Idee war, hierzu auf einer „Index-Seite“ sämtliche aktuellen Projekte in Form einer Tabelle darzustellen und hinter jedem Projektnamen den Link auf die entsprechende Web-Page zu hinterlegen.

Die letzte offene Frage ist nun noch die Realisierung des Gesamttests eines jeden Projektes. Denn selbst wenn alle einzelnen NE korrekt funktionieren, bedeutet dies nicht, dass das Zusammenspiel zwischen den Netzelementen ebenfalls fehlerfrei abläuft und die reibungslose End-to-End-Kommunikation im Sinne eines Mobilfunknetzes möglich ist.

Für solche End-to-End-Tests (Projekttests) wird ein bereits von Mitarbeitern der Siemens AG entwickeltes Tool Namens „Independent Protocol Simulator“ (kurz: „IPS“) verwendet. Nach entsprechender manueller Eingabe aller projektspezifischen Informationen startet dieses Tool verschiedenste Testszenarien (im Folgenden auch Testrufe oder Testcalls genannt) und nimmt eine Auswertung vor. Diese Auswertung wird in Form einer Textdatei gespeichert. Die Idee war nun, das IPS-Tool in die eigene Anwendung einzubinden und die Übergabe der projektspezifischen Informationen zu automatisieren. Dem Anwender wird so die Möglichkeit gegeben, „On Demand“ vorher definierte Testszenarien auszuführen. Die angezeigten Ergebnisse wären somit immer auf dem aktuellen Stand. Realisiert werden könnte dies über Buttons auf den jeweiligen Web-Seiten der Projekte. Jeder Button steht dabei für ein Testszenario. Wird ein Button angeklickt, wird das Szenario im IPS abgearbeitet, ausgewertet und das Ergebnis auf der entsprechenden Projektwebseite visualisiert.

Die in Abbildung 12 als „Web-Application“ bezeichnete Stufe der Anwendungsentwicklung beschäftigt sich mit genau diesen Punkten.

Einen Ausschnitt aus einem IPS-Protokol zeigt die folgende Abbildung 13.

.

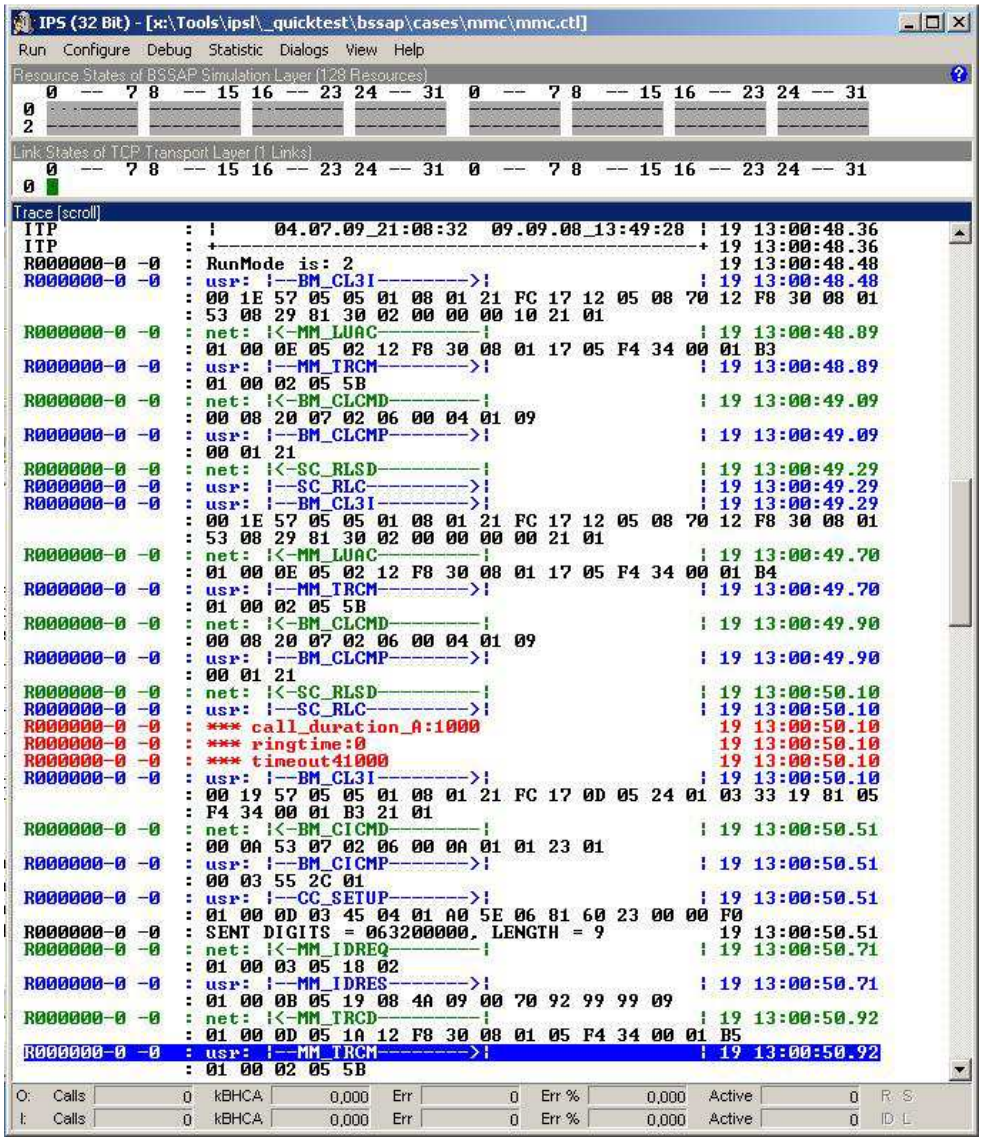

Abbildung 13: Beispiel IPS-Test

Zusammenfassend betrachtet besteht die zu entwickelnde Anwendung also aus zwei Teilen:

1. Einem statischen Teil (ENC– Easy Net Checker → siehe Kapitel 6)
    a. Ermittlung der Zustände und Zusatzinformationen der NE
    b. Ermittlung der jeweiligen NE eines jeden aktuellen Projektes
    c. Erstellen entsprechender Übersichtsbilder eines jeden Projektes mit aktuellen Zuständen der NE
    d. Generieren der Web-Pages der einzelnen Projekte

2. Einem dynamischen Teil (TCA – Test Call Analyzer → siehe Kapitel 6)
    a. Auslösen eines Testcall-Szenarios auf Knopfdruck
    b. Aktualisierung der Web-Page mit Ergebnissen der Test Call Analyse

Bei der Entwicklung sind weiterhin folgende Punkte zu beachten:

- Die zu entwickelnde Lösung soll übersichtlich und kurz den Zustand eines Projektes und dessen einzelner Netzelemente wiedergeben. Es ist dabei nicht Sinn und Zweck, dem Bedienpersonal tiefer gehende Informationen zum Projekt zu liefern.
- Die Schnittstellendefinition zwischen den Netzelementen und dem „Interface Agent“ aus Abbildung 11 ist festzulegen. Dabei ist nur zu definieren, wie das Ergebnis vorzuliegen hat. Nach welchen Kriterien das Ergebnis ermittelt wird, ist Aufgabe des Netzelementebetreuers.
- Nutzen bereits vorhandender Teillösungen.
    - Zu jedem Projekt werden in der Arbeitsgruppe bereits verschiedene Dateien erstellt, die für die Auswertung als Quellen verwendet und nicht selbst erstellt werden müssen.
    - Des Weiteren existiert ein Programm für die Analyse verschiedener Testszenarien, welches für die Realisierung des dynamischen Teils genutzt werden kann.

4.2 Vor- und Nachteile des ENC/TCA

Vorteile	Nachteile
• Projektspezifische Ansicht relevanter Netzelemente • Projektzustand „On Demand“ testbar • Allgemeine Schnittstellendefinition • Hohe Flexibilität bei der Zustandsbestimmung der NE • Einfaches Filesystem als Datenbank	• Plattformabhängig • Abhängig von Angaben der NE-Betreuer • Abhängig von Zugaben aus anderen Programmen

Tabelle 2: Vor- und Nachteile ENC/TCA

Die meisten vorgestellten Tools überwachen einzelne Dienste von Netzwerkkomponenten. Nagios überwacht z. B. SMTP und POP3, BigBrother überwacht die TCP Dienste, MRTG analysiert hauptsächlich die Auslastung von Servern und Routern. Die Verbindungen und damit die Kommunikation zwischen den einzelnen Elementen werden bei BigBrother und Xymon überwacht. Allerdings wird auch bei diesen beiden zuletzt genannten Tools die Aktualisierung der Zustände immer in einem definierten Zeitintervall durchgeführt. Außerdem gibt es keine Möglichkeit, die überwachten Netzwerkkomponenten zu definierten Projekten zusammenzusetzen. Der Benutzer sieht also stets alle überwachten Elemente eines Netzwerkes.

Mithilfe des Easy Net Checkers wird nun, anders als bei den vorgestellten Tools, die Möglichkeit gegeben, alle überwachten Netzkomponenten projektspezifisch zugeordnet zu betrachten und deren Zustände anzeigen zu lassen. Der Benutzer wird somit nicht mit den Zuständen aller NE überhäuft, sondern er erhält durch Auswahl eines Projektes nur die Zustandsinformationen der Komponenten des gewählten Projektes.

Durch die Entwicklung einer allgemein gültigen Schnittstelle zwischen den einzelnen Netzkomponenten und dem ENC/TCA wird es außerdem möglich, Netzelemente unabhängig von dessen Anbindung an ein Netzwerk zu überwachen. Durch diese Schnittstelle ist es dem Nutzer zudem möglich, schnell und unkompliziert Netzelemente aus der Überwachung zu entfernen oder hinzuzufügen.

Ein weiterer Vorteil der Software entsteht durch die Flexibilität bei den Zustandsbestimmungen der einzelnen Netzelemente. Es sind keine festen Regeln und Tests definiert, stattdessen entscheiden die entsprechenden NE-Betreuer welche Faktoren Einfluss auf die Zustandsdefinition eines NE haben. Diese Faktoren können von den Betreuern jederzeit angepasst werden, ohne dabei direkte Änderungen im ENC/TCA vornehmen zu müssen.

Der größte Vorteil ist allerdings die Möglichkeit „On Demand“, auf Knopfdruck, ein Projekt auf seine Funktionalität testen zu können. Der Tester kann direkt vor dem Start seines komplexen Tests die Grundfunktionalität des gewählten Projektes feststellen und so die Möglichkeit umgehen, dass eine Funktionsbeeinträchtigung zwischen 2 Abfrageintervallen unbemerkt bleibt.

Da die Entwicklungsumgebung auf Windows-Betriebssystemen basiert, ist die Plattformabhängigkeit der vorliegenden Software als Nachteil anzusehen. Durch Verwendung entsprechender Compiler ist es allerdings möglich, dieses Tool auch auf anderen Betriebssystemen einzusetzen. Für die Verwendung in der Arbeitsgruppe ist dies allerdings irrelevant, da sämtliche Desktop PC’s mit Windows Betriebssystemen arbeiten und auch in Zukunft arbeiten werden.

Die eingeräumten Freiheiten für die NE-Betreuer bei der Bestimmung der Zustandsdefinitionen beeinflussen direkt die Aussagekraft des Programmes. Eventuell unnötige oder sogar fehlerprovozierende Deklarationen können allerdings auch ohne direkten Eingriff in den Quellcode der Anwendung wieder behoben werden. Besonders der Test Call Analyzer ist auf Dateien des IPS-Tools angewiesen, welche zur Ausführung der Testrufe notwendig sind. Werden diese Dateien in Zukunft nicht weiter gepflegt, muss das Programm entsprechend angepasst werden.

5 Entwicklungsbeschreibung der Software

Zu Beginn der Lösungsbeschreibung wird kurz auf das mit einbezogene IPS-Tool eingegangen. Danach wird die Infrastruktur, in der die Anwendung arbeiten soll, näher erläutert. Daraufhin erfolgt die Beschreibung der Schnittstelle zwischen den NE und der eigentlichen Anwendungen. Als weitere Grundlage der Lösungsbeschreibung erfolgt nun die Erläuterung der verwendeten Main.ini als zentrale Sammelstelle vieler nötiger Definitionen für die beiden Anwendungsteile ENC und TCA. An die Beschreibung der Main.ini schließt sich dann der statische Teil, der Easy Net Checker, an, bevor es dann im letzten Abschnitt dieses Kapitels um den generischen Teil, den Test Call Analyzer gehen wird.

5.1 Independent-Protocol-Simulator-TOOL

IPS ist ein Testtool, entwickelt von Mitarbeitern der Firma Nokia Siemens Networks, mit dem es möglich ist, die Kommunikation verschiedenster Protokolle (z. B. SS7[43] oder IP) und deren OSI[44]-Schichten zu testen. Das IPS-Tool basiert auf Microsoft Windows. Durch die gegebene Benutzeroberfläche wird das Nachvollziehen eines Testprozesses möglich. Während der Bearbeitung eines Testfalles wird zusätzlich noch eine Zusammenfassung des aktuellen Stands des Tests generiert (TRC-File[45]). Die Abarbeitung paralleler Testfälle wird unterstützt und ermöglicht so auch das Testen der Kommunikation zwischen Testfällen (TCP/IP oder RS232, bzw. V.24). Ein IPS-Testfall wird über ein Konfigurationsfile gestartet („*.cfg“). In diesem File stehen fest definierte sowie projektspezifische Einstellungen. Dieses Tool wird für die Realisierung der Systemtests im TCA verwendet.

[43] Signalling System Number 7
[44] Open Systems Interconnection
[45] Trace File

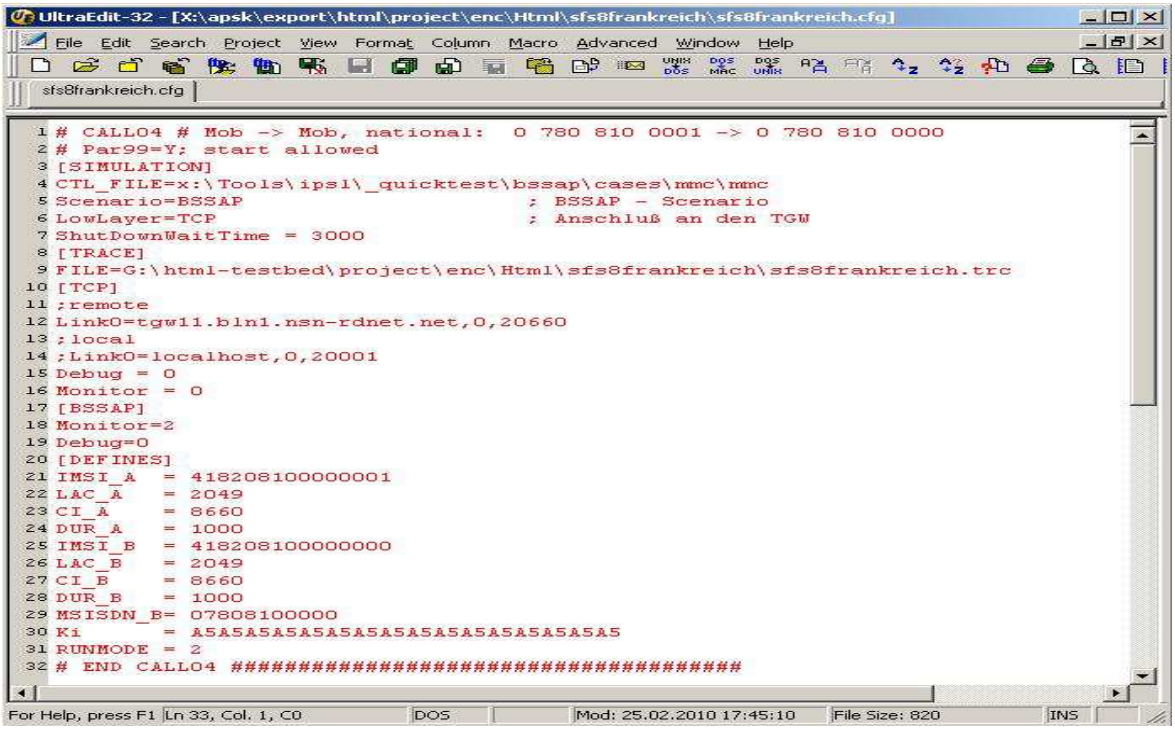

Abbildung 14: Beispiel einer CFG-Datei

Abbildung 14 zeigt einen Ausschnitt einer CFG-Datei. Die erste Zeile beschreibt dabei die Art des Calls und die verwendeten Testrufnummern. Hierbei handelt es sich um einen Testruf von einem Mobilteilnehmer zu einem zweiten („Mob->Mob“). Die entsprechenden Informationen für einen solchen Call erhält das IPS-Tool über ein externes CTL-File (siehe Zeile 4). In welchem Testsystem der Call ausgeführt werden soll, wird in Zeile 9 festgelegt. Im Folgenden werden noch weitere Informationen über das Testnetz und die verwendeten Mobilteilnehmer gesammelt, auf die an dieser Stelle aber nicht weiter eingegangen werden soll.

5.2 Infrastruktur der Entwicklungsumgebung

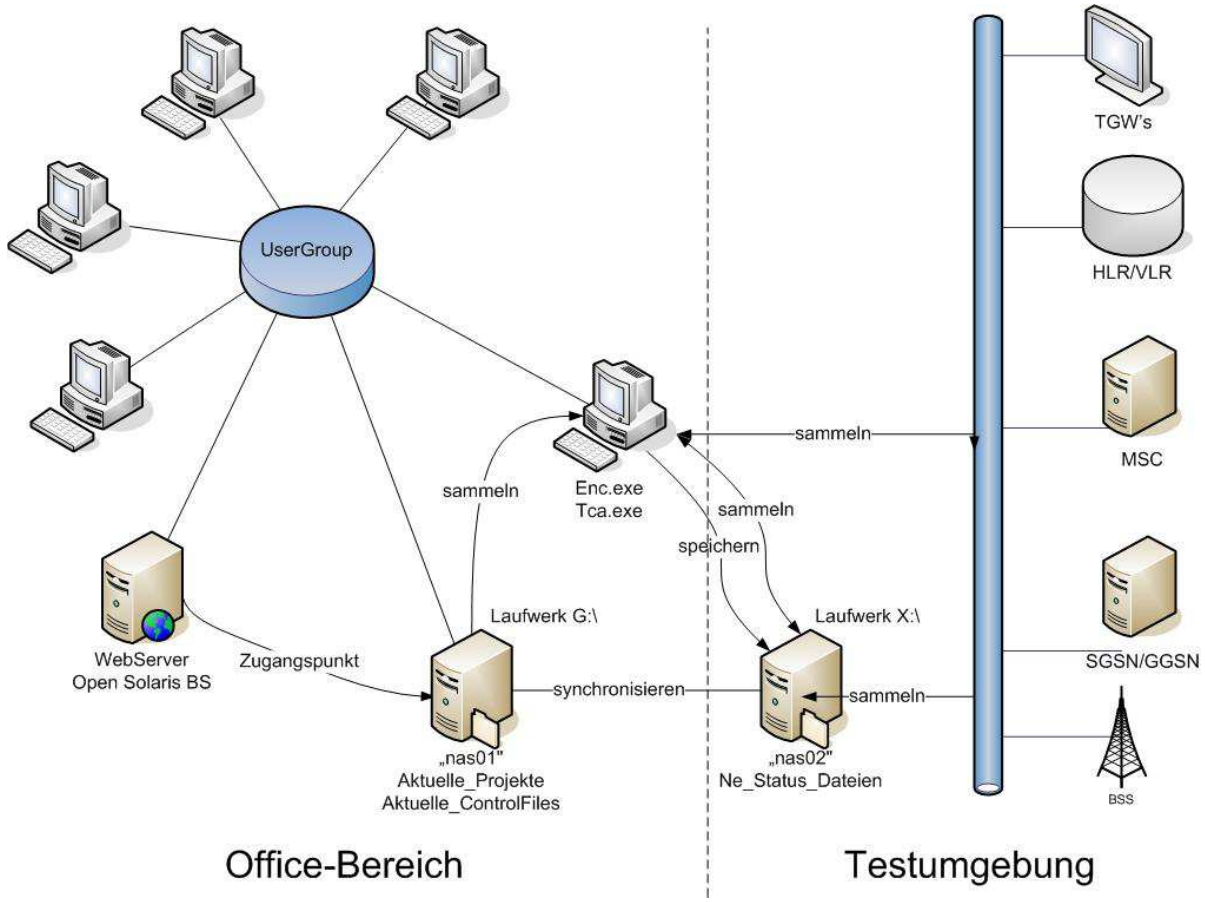

Abbildung 15: Infrastruktur der Entwicklungsumgebung

Das Arbeitsumfeld gliedert sich in zwei Teile (siehe Abbildung 15), einerseits den Teil des Office-Bereiches und andererseits den Teil der Testumgebung. Der Office-Bereich beschreibt dabei den normalen „Büroalltag". Dazu gehören E-Mails lesen und schreiben, Dokumente anfertigen, Skripte und andere Programme erstellen, Daten speichern, usw. Der zweite Bereich ist die Testumgebung. In diesem Bereich befinden sich die IP-Netze der Netzelemente. Die Betreuer der NE haben entweder direkt oder über entsprechende Verwaltungstools Zugriff auf die Testumgebung und somit auf die Netzelemente. Über diese Schnittstelle zwischen Betreuer und Netzelement werden das Ausführen von Testszenarien und das anschließende Auswerten verschiedenster Testfälle ermöglicht. Des Weiteren ist der Zugriff auf einen Web-Server gegeben, der es der Arbeitsgruppe ermöglicht, Ergebnisse zu präsentieren und auch Gruppenübergreifende Informationen zu visualisieren.

Die Daten des ENC und des TCA werden an einer zentralen Stelle gesammelt und in folgende Unterverzeichnisse gegliedert:

- …/Html
- …/Projekte
- …/NE

Im „Html“ Ordner werden alle Web-Seiten der einzelnen Projekte abgespeichert. Die Projektdateien, mit ihren jeweils zugehörigen Netzelementen, werden als Textdatei im „Projekte“ Ordner abgelegt. Der „NE“ Ordner beinhaltet alle Zustandsdateien der Netzelemente und bildet die Schnittstelle zwischen Netzelementen und dem „Easy Net Checker“.

5.3 Schnittstellenbeschreibung

Um die Zustände aller Netzelemente sinnvoll verwalten und auswerten zu können, benötigt man eine einheitliche Schnittstelle zwischen den NE‘s und dem Programm, welches die Zustände projektspezifisch auswertet, dem „Easy Net Checker“ (siehe Lösungsansatz Kapitel 4). Als Schnittstelle dient das Verzeichnis „…\NE“, indem sämtliche Zustandsdateien der einzelnen Netzelemente abgelegt und gespeichert werden. Diese Zustandsdateien haben ein definiertes Format, wie in Abbildung 16 veranschaulicht wird.

Abbildung 16: Zustandsdatei der Netzelemente

In jeder NE-Datei müssen sich folgende Schlüsselwörter wiederfinden, um eine erfolgreiche Auswertung zu ermöglichen: “[STATUS]“, “[/STATUS]“. Zwischen diese beiden Schlüsselwörter ist jeweils der ermittelte Zustand zu hinterlegen. Wie dieser Zustand ermittelt wird und wie diese Datei angelegt wird, bleibt dem verantwortlichen Mitarbeiter/Netzelementebetreuer überlassen. Er erstellt eine jeweils an sein Netzelement angepasste Routine (Skript) mit allen für ihn relevanten Funktionstests und anschließend die NE-Datei mit dem entsprechend ermitteltem Zustand. Diese Herangehensweise ermöglicht es den Mitarbeitern der Arbeitsgruppe, jederzeit individuell Netzelemente in die Verwaltung aufzunehmen oder andere herauszulöschen, ohne am eigentlichen Programm Änderungen vornehmen zu müssen.

Als Zusatzinformationen zu einem Netzelement wurden weitere Schlüsselwörter definiert:

- „[HOST]“ und „[/HOST]“

  Standort des NE im Netz (IP Adresse).

- „[AGENT]“ und „[/AGENT]“

  Falls ein Netzelement über ein anderes Tool verwaltet wird, wie zum Beispiel das MSC über den „Switch Commander“, können Informationen (z.B. IP Adresse) zu dem entsprechenden Agent zwischen diese beiden Schlüsselwörter definiert werden.

- „[LOGFILE]“ und „[/LOGFILE]“

  Hier befindet sich gegebenenfalls der Pfad zu dem entsprechenden Logfile des ausgewerteten NE. Logfiles sind bei eventuellen Fehlern ein erster Anlaufpunkt für die Tester, um möglichst schnell Fehler zu finden und korrigieren zu können.

- „[CONTACT]“ und „[/CONTACT]“

  Diese beiden Schlüsselwörter werden verwendet um einen Ansprechpartner bei eventuellen Fragen zu definieren. Angaben zur E-Mail Adresse und Telefonnummer sind hier vermerkt.

Die entscheidende Information ist hierbei die Statusmeldung. Alle anderen Informationen sind für die Funktion der entwickelten Anwendung vorerst nicht von größerer Bedeutung. Allerdings liefern sie einen ersten Anreiz zur späteren Erweiterung (siehe Kapitel 6).

Die erzeugten Files und das Verzeichnis bilden die Schnittstelle zwischen den Netzelementen und dem ENC und TCA.

5.4 Initialisierungsdatei

Neben der Schnittstelle ist die Definition einer Main.ini ebenfalls eine entscheidende Grundlage für die beiden Teilprogramme ENC und TCA. Die hier gesammelten Daten werden von beiden Teilprogrammen verwendet. Änderungen wie z. B. Pfadangaben zu Quell- oder Zielverzeichnissen können von den Benutzern in der Main.ini vorgenommen werden. Die neuen Daten werden beim nächsten Aufruf der Programme übernommen, ohne dass man in den Quellcode direkt eingreifen muss. Auf diese Weise ermöglicht man dem Benutzer eine einfache Anpassung der Anwendung an eventuell aufgetretene Änderungen. Die folgende Abbildung 17 zeigt die verwendete Initialisierungsdatei „Main.ini“.

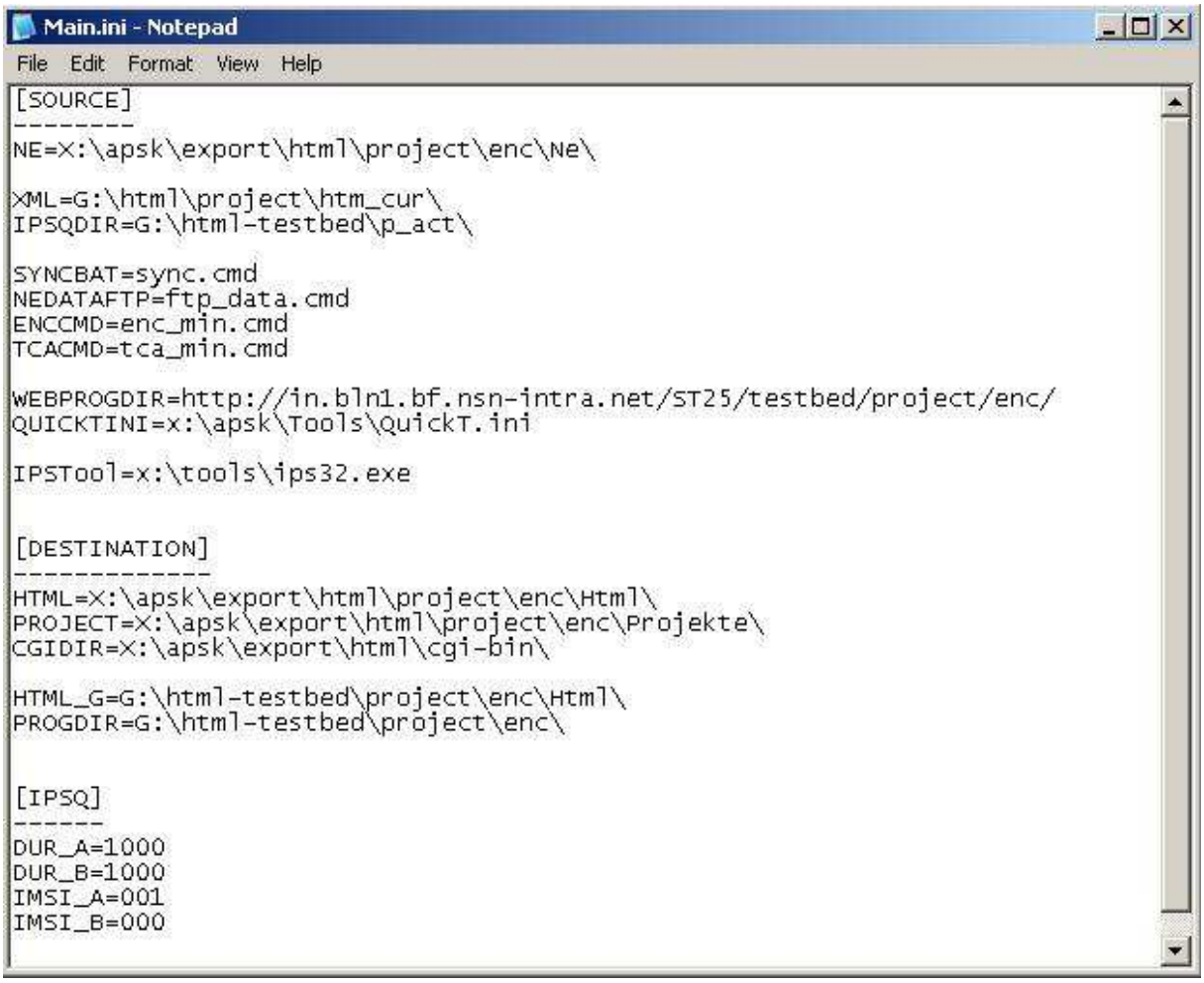

Abbildung 17: Main.ini

Unter der Sektion „SOURCE“ sind in der „Main.ini“ alle Quellverzeichnisse und Verzeichnisse der zusätzlich verwendeten Software Tools hinterlegt, die im Programm Verwendung finden.

“NE“ bezeichnet dabei das Verzeichnis, indem die Zustandsdateien der Netzelemente abgelegt werden. Hinter „XML“ wird der Pfad zu allen aktuellen Projekten hinterlegt. In jedem Projektordner existiert ein XML-File aus dem der ENC zu jedem Projekt die entsprechenden NE herausfiltert. Die Verzeichnisse, die sich hinter „NE“ und „XML“ verbergen, bilden die Datenbasis der entwickelten Software.

Für die Realisierung der Systemtests[46] ist hinter dem Schlüsselwort „IPSQDIR“ das Quellverzeichnis vermerkt, indem sich alle aktuellen Projekte mit deren Konfigurationsfiles befinden.

[46] Systemtest – Test eines gesamten Projektes. Auswertung der Antworten von definierten Testrufen

Die nun folgenden Batch-Files (siehe Abbildung 17) dienen der Synchronisation im firmeninternen Netzwerk und der Minimierung der Teilprogramme während der Ausführung.

Aus der Zeile „WEBPROGDIR“ geht der Link hervor, indem sich die HTML-Übersichtsseite aller aktuellen Projekte befindet.

Anschließend wird der Verzeichnispfad der QuickTIni-Datei angegeben. Die QuickTIni enthält die Grundgerüste einiger Test-Call-Szenarien, die für die Ausführung des IPS-Tools und somit für den TCA wichtig sind.

Neben den Quellen werden in der Main.ini auch Zielpfade (Sektion: „DESTINATION“) definiert. Zielpfade sind Speicherorte für die Daten, die während der Programmausführung generiert werden. Dazu gehören:

- HTML: Projektspezifische HTML-Files für die Visualisierung
- PROJECT: Projektdateien mit den dazugehörigen Netzelementen
- CGIDIR: Skripte, die auf den Web-Seiten verwendet werden können
- HTML_G: Verzeichnis für HTML-Dateien auf Laufwerk „G“
- PROGDIR: Verzeichnis des Programms auf Laufwerk „G“

Die letzten beiden genannten Verzeichnisse sind wegen unterschiedlicher Zugriffsrechte und entsprechend notwendiger Synchronisationen im firmeninternen Netzwerk notwendig.

Als letzten Teil der Main.ini wurden in der Sektion „IPSQ“ einige Daten eingetragen, die die Testrufe beeinflussen, z. B. die Dauer der Rufe und die Endekennziffern (die ersten Ziffern sind immer identisch) der IMSI‘s der eingerichteten Testteilnehmer in den Testanlagen.

Sollten sich im Laufe der Nutzung des Tools die Quellenverzeichnisse oder andere hier hinterlegte Daten ändern, reicht ein Editieren dieser Datei aus, um das Programm an die neuen Bedingungen anzupassen.

5.5 Entwickelte Software ENC und TCA

5.5.1 Entwicklung ENC

Der Easy Net Checker realisiert, wie schon in Kapitel 4 erwähnt, den statischen Teil der Anwendung.

Dazu zählt das Auslesen der Zustände der Netzelemente aus den angelegten NE-Dateien, die Zuordnung der NE zu den aktuellen Projekten, das Erstellen des Übersichtsbildes, sowie die Generierung der entsprechenden Web Seiten zu jedem Projekt.

Die Hauptaufgabe des Easy Net Checker (kurz:"ENC") ist die projektspezifische Analyse der NE-Dateien und die Realisierung der Visualisierung der ermittelten Ergebnisse. Bevor man mit der Projektspezifischen Analyse beginnen kann muss zuerst geregelt werden, wie dem Anwendungsprogramm alle Netzelementedateien zur Verfügung gestellt werden sollen. Eine Möglichkeit war die Objektorientierte Herangehensweise. Dabei wird eine Klasse „NE“ angelegt und mit entsprechenden Attributen, wie zum Beispiel: NE-Zustand, ausgestattet. Alle Netzelemente werden nun als Objekte der Klasse „NE“ hinzugefügt.

Eine ähnliche, aber dabei weniger rechenlastige Variante ist die Verwendung einer „Array of Record“-Struktur, welche bei der Realisierung des ENC aus genau diesem Grund auch verwendet wurde. Die Funktionsweise wird im folgenden Kapitel noch erläutert.

Es folgte die Problemstellung der Ermittlung der projektspezifischen NE. Hierzu gab es zu Beginn der Entwicklung die Idee, aus dem in der Arbeitsgruppe vorhandenen Visio Bild, welches zu jedem Projekt angelegt wird, die verwendeten NE herauszufiltern. Die NE des jeweiligen Projektes sollten dann auf ihren Zustand überprüft werden und ihrem Zustand entsprechend mit einer bestimmten Farbe direkt im Visio Bild dargestellt werden. Hieraus ergaben sich 2 Probleme:

- Die Visio-Dateien konnten in ihrem allgemeinen Speicherformat (Dateien mit der Endung *.vsx) nicht von Delphi verwendet werden.
- Farbliche Änderungen am Bild waren somit auch nicht direkt möglich

Microsoft Visio bietet allerdings die Möglichkeit, angelegte Projekte im HTML-Format zu speichern. Auf diese Weise wird das Übersichtsbild als GIF[47]-Datei gespeichert und alle Textinformationen in einer XML[48]-Datei hinterlegt. Somit war es nun möglich aus der XML-

[47] Graphics Interchange Format
[48] Extensible Markup Language

Datei die NE des entsprechenden Projektes herauszufiltern und in einer separaten Textdatei (im folgenden Projektdatei genannt) abzulegen. Die geplante farbliche Anpassung der NE direkt im Übersichtsbild war im GIF-Format nicht möglich. Als Alternative entstand die Realisierung einer Tabelle neben dem Übersichtsbild, in der sämtliche NE des Projektes mit ihren entsprechenden Zuständen aufgelistet werden sollten. Die Zustände sollten dabei mit einer entsprechenden Hintergrundfarbe hervorgehoben werden.

Der ENC wurde zu Beginn der Programmierarbeiten darauf ausgelegt, einmal am Tag gestartet zu werden. Er sollte in einer Zeitschleife alle Funktionen und Prozeduren wiederholt abarbeiten und so, je nach Größe der Zeitspanne, aktuelle Informationen bereitstellen. Der Nachteil dabei ist, dass das Programm die ganze Zeit aktiv ist und Ressourcen des Desktop-PC's benötigt. Hinzu kommt, dass bei einem Programmabsturz im Fehlerfall das Programm bis zu einem manuellen Neustart durch den Benutzer nicht mehr aktiv wäre. Eine bessere Variante ist es, die Schleifenfunktion vom Betriebssystem übernehmen zu lassen (in diesem Falle, dem Windows-Scheduler – siehe Abbildung 18). Der Benutzer des ENC kann somit einfach und elegant die Zeitspanne und die Frequenz der Wiederholungen der Anwendung definieren und gegebenenfalls anpassen. Des Weiteren ist im Fehlerfall nur ein Durchlauf des Programmes betroffen.

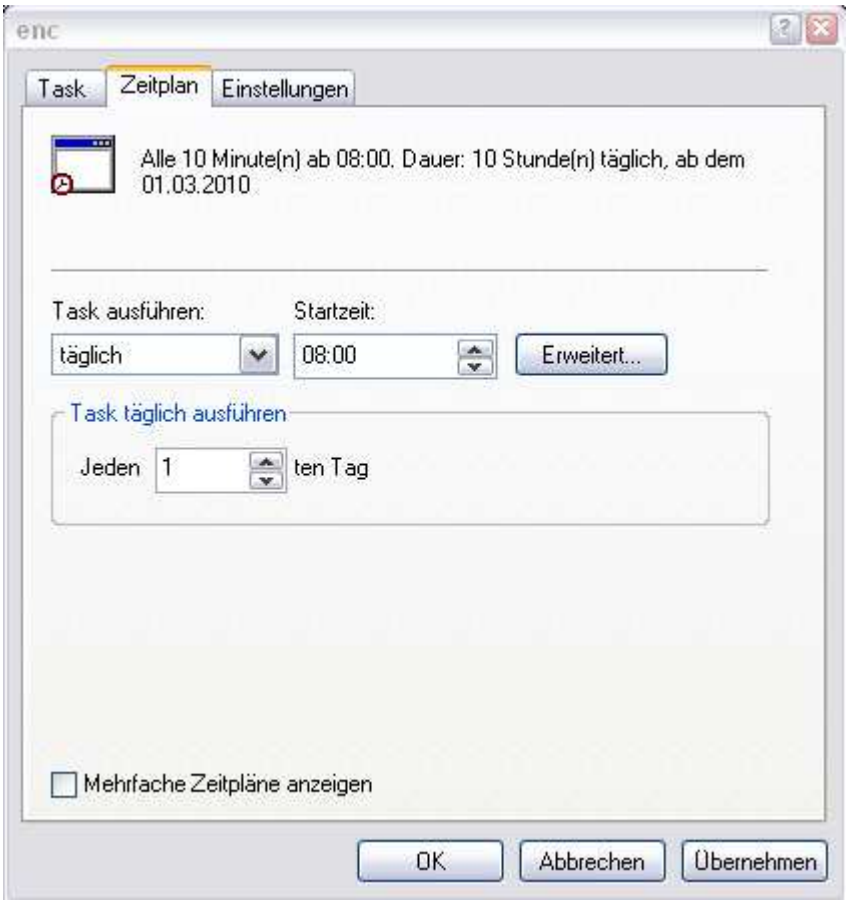

Abbildung 18: Windows Scheduler

Die Abbildung 18 zeigt die aktuellen Einstellungen des Windows Scheduler. Der Scheduler startet also ab 8:00 Uhr alle 10 Minuten den Easy Net Checker. Der Nutzer erhält also beim Aufruf eines Projektes immer Datensätze die maximal 10 Minuten alt sind.

5.5.2 Programmablauf ENC

Die Abbildung 19 zeigt den Programmablaufplan des ENC, der im Folgenden näher erläutert werden soll.

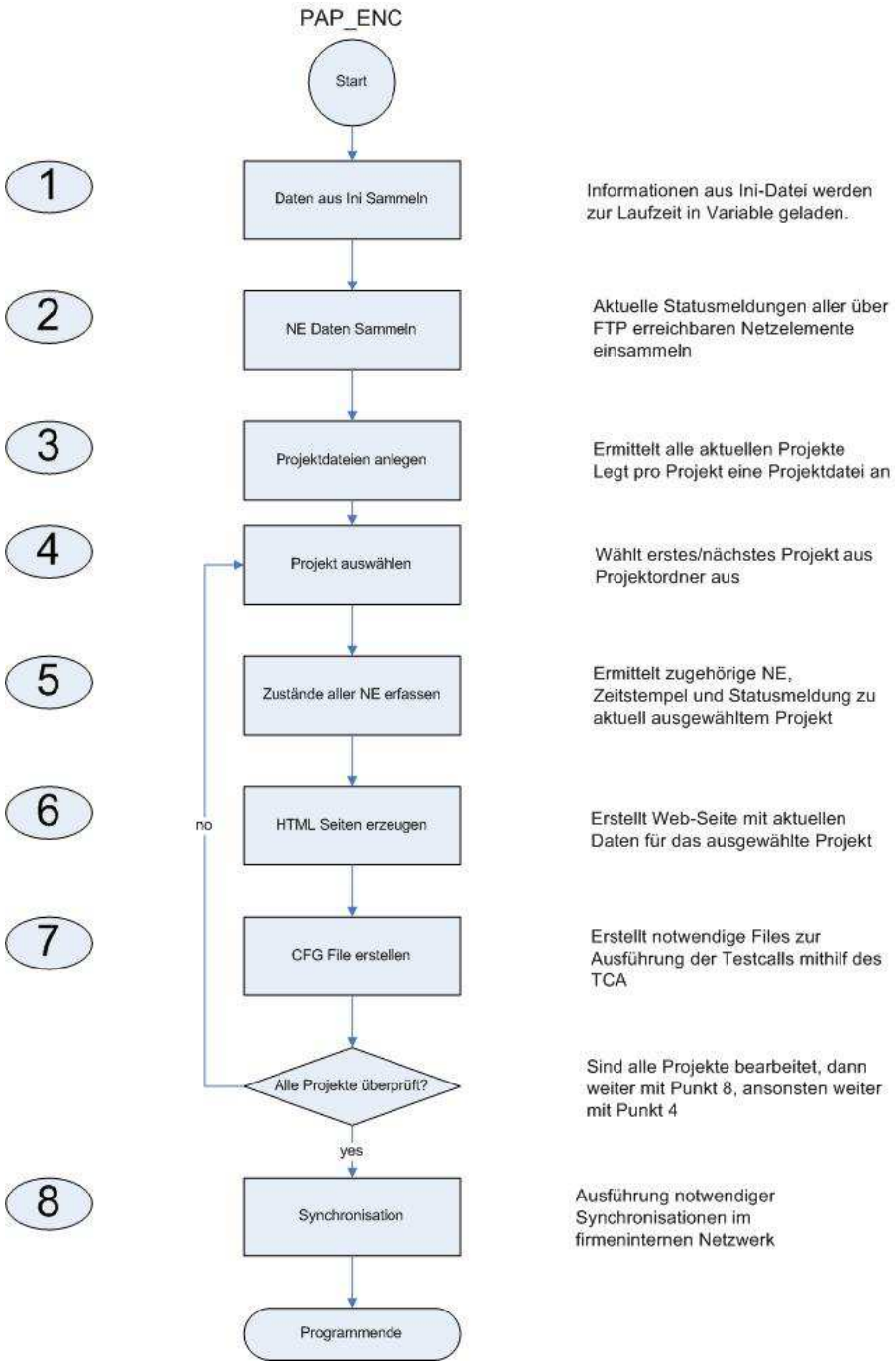

Abbildung 19: PAP_ENC

1 Daten aus INI Sammeln

Nach dem Start des Programms wird mithilfe der Prozedur „MyIniDataCollection“ die „Main.ini“ ausgelesen und alle für den ENC relevanten Werte werden in definierte Variable geladen. Die folgenden Quelltexte zeigen den Aufbau des „arrays“ in dem die „Main.ini“ Informationen während der Laufzeit des Programmes gespeichert werden und die Prozedur „MyIniDataCollection“.

```
Type TIniEintrag = record
        Ne : string; //Ordner Netzelemente
        Projekt : string; //Ordner Projekte
        Log : string; //Ordner LogFiles
        Html : string; //Ordner Web-Seiten
        VisioHtml : string; //Ordner Visio XML Datei
        CFGFile : string; //Ordner Config File

End;
```

Es muss sichergestellt werden, dass die Ressource „Main.ini“ auch beim Auftreten einer Exception wieder freigegeben wird. Durch die Verwendung einer „try…finally-Anweisung“ wird dies realisiert. / (Delphi Treff: Fehlerbehandlung 2010)/

```
procedure MyIniDataCollection;
var
ini : TMemIniFile;
begin
ini:=TMemIniFile.create(ParamStr(1));
try
IniData.Ne := ini.ReadString('SOURCE','NE','');
IniData.XML := ini.ReadString('SOURCE','XML','');
…
finally
ini.free;
end;

end;
```

2 NE Daten sammeln

Netzelemente, die über FTP[49] erreichbar sind, werden vom ENC in den definierten NE-Ordner kopiert (siehe Kapitel 5.3 Schnittstellenbeschreibung).

Dies erfolgt über den Aufruf der Prozedur „ExecuteIt“, welche ihrerseits die ausführbare Datei „ftp.cmd“ startet.

```
ExecuteIt(IniData.NEDATAFTP,true,true,true,true);
```

Die erste übergebene Variable enthält die FTP-Kommando-Datei. Die folgenden vier Werte sind Wahrheitswerte und beeinflussen die Ausführung der CMD-Datei.

Die Übergabe von „True“ oder „False“ beeinflusst in der Reihenfolge folgende Ausführungseigenschaften:

[49] File Transfer Protocol

- Separates Konsolenfenster starten
- Fenster minimiert ausführen
- Datei im Hintergrund ausführen
- Programmablauf anhalten bis Exe-Datei beendet

Die Kommandodatei wird also in einem separaten Fenster, minimiert und im Hintergrund gestartet. Der ENC arbeitet erst weiter, nachdem die „ftp.cmd“ abgearbeitet wurde.

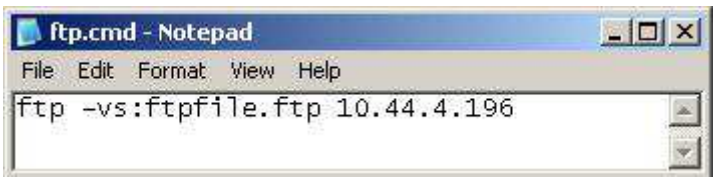

Abbildung 20: FTP Command File

Die Kommandodatei aus Abbildung 20 startet den FTP-Server. Notwendige Daten der Quell- und Zielordner, sowie die Informationen zur Nutzeranmeldung erhält die Kommandodatei von einem „*.ftp-File“.

Es sei an dieser Stelle erwähnt, dass alle NE die nicht per FTP erreichbar sind, durch individuelle Tools der jeweiligen Netzelementebetreuer regelmäßig abgefragt und die entsprechenden Statusdateien (NE-Dateien) im dafür vorgesehenen Ordner abgelegt werden.

Bevor nun die eigentliche Zustandsauswertung erfolgt, werden zuerst sämtliche Ordner geleert, die eventuell Dateien enthalten, die während früherer Ausführungen des ENC entstanden sind (Projekt-, HTML- und Skript-Ordner). So wird verhindert, dass veraltete Dateien das Ergebnis beeinflussen (siehe folgenden Quelltext).

```
DelFiles(IniData.Project+'*.*',true,false); //löscht alle erzeugten ProjectFiles auf X
DelFiles(IniData.Html+'*.*',true,false); //löscht alle erzeugten Html Files auf X
DelFiles(IniData.Html_G+'*.*',true,false); //löscht alle erzeugten Html Files auf G
DelFiles(IniData.ProgDir+'Projekte\*.*',true,false);//löscht alle erzeugten ProjectFiles auf G
DelFiles(IniData.CGI_BIN+'*.*',true,false); //löscht alle erzeugten txt Files im cgi bin or
```

3 Projektdateien anlegen

Mithilfe der Funktionen „FindFirst“ und „FindNext“ sowie einer While-Schleife werden, in der Prozedur „ReadXmlFiles“ (siehe Seite 48), die Projekte im entsprechenden Projektordner („G:\html\project\htm_cur\“) sequentiell durchsucht und das jeweilige XML-File aus den gegebenen HTML-Ordnern der Projekte ermittelt. Das entsprechende XML-File wird dann mit Lesezugriff geöffnet („OpenTextForRead“). Anschließend wird im Projektordner („X:\apsk\export\html\project\enc\Projekte\“) eine Textdatei mit dem Namen des aktuellen Projektes erstellt und mit Schreibrechten geöffnet („MyOpenTextForWrite“).

Die Prozedur „CollectXMLData“ realisiert im nächsten Schritt die Ermittlung der Netzelemente, die zu einem Projekt gehören. Dabei wird das jeweilige XML-File Zeile für Zeile nach dem Schlüsselwort „NE::“ durchsucht und alle gefundenen Werte werden in die geöffnete Textdatei im Projektordner geschrieben. Dabei wird in jede Zeile nur ein Eintrag geschrieben. Nachdem die Unterprozedur „CollectXMLData“ beendet wurde, werden mithilfe von „CloseText“ und „MyCloseText“ auch die XML-Datei und die Projektdatei geschlossen.

```
procedure ReadXmlFiles(VisioDir,ProjectDir:string)
var entry : TSearchRec;
ProjectName : string;
FileSearch : Integer;
FileToRead,FileToWrite : text;
begin
FileSearch:=FindFirst(VisioDir+'*.htm' ,faAnyFile,entry); //Pointer auf File im N
while FileSearch = 0 do
begin
ProjectName := ExtractFileNameWithoutExt(entry.name); //filename ohne .*
if FileExists(VisioDir+ProjectName+'_files\data.xml') then //Englische Visio Version
OpenTextForRead(VisioDir+ProjectName+'_files\data.xml',FileToRead,1024) //Pfadangabe + Textdatei
else
OpenTextForRead(VisioDir+ProjectName+'-Dateien\data.xml',FileToRead,1024); //Deutsche Visio Version
MyOpenTextForWrite(FileToWrite,ProjectDir+ProjectName+'.txt');
CollectXmlData(FileToRead,FileToWrite,'NE::','</Text>',4,7);
CloseText(FileToRead);
MyCloseText(FileToWrite);
FileSearch:=FindNext(entry);
end;
FindClose(entry);

end;
```

4 Projekt auswählen

Nachdem man jetzt die zugehörigen Netzelemente zu jedem Projekt ermittelt hat, erfolgt die projektspezifische Abarbeitung und Auswertung der Zustände der Netzelemente.

Dazu wird, ähnlich wie in der Prozedur „ReadXMLFiles“, eine „While-Schleife“ verwendet. Mithilfe der Funktionen „FindFirst“ und „FindNext“ werden nacheinander die Projektdateien im Projektordner geöffnet und jeweils folgende Prozeduren abgearbeitet:

- „GetNeNames“
- „ProofTimestamp“
- „NeDataCollect“
- „MakeHtmlFiles“
- „GenerateCFG“

Die Ergebnisse dieser Prozeduren werden in einem array-of-record gespeichert, welches sich aus folgenden Datenfeldern zusammensetzt:

Name	String	Name des Netzelementes
Status	String	Zustandsmeldung des NE
Host	String	Zusatzinfos
LogFile	String	Zusatzinfos
Contact	String	Zusatzinfos
TimeOK	Longword	Datum in Sekunden seit 01.01.1970
Value	Boolean	Legt fest, ob ein NE ausgewertet wird, oder nicht.

Tabelle 3: Array Netzelemente

5 Zustände aller NE erfassen

Durch die Zeilenweise Speicherung der NE in der Prozedur „ReadXMLFiles“ ist es nun möglich in der Prozedur „GetNeNames“ (siehe Quelltext am Seitenende) die Projektdateien als String-Liste zu laden. Die Namen der NE des gerade aktiven Projektes (Projekt, welches im Moment in der While Schleife bearbeitet wird) werden jeweils in das Datenfeld „Name“ eingetragen.

Die Funktion „ListLength“ begrenzt die Anzahl der Array Einträge auf die Anzahl der NE des aktiven Projektes. Somit wird ausgeschlossen, dass mehr Netzelemente einem Projekt zugeordnet werden als tatsächlich vorhanden sind.

```
Procedure GetNeNames(ProjectDir,ActiveProject:string);
var
i,j : integer;
list:TStringList;
begin
list:=TStringList.Create;
list.LoadFromFile(ProjectDir+ActiveProject);
list.Sort;
i:=0;
try
for j:=0 to ListLength (ProjectDir,ActiveProject)-1 do
begin
Ne[i].Name:=list.Strings[j];
if i< ListLength (ProjectDir,ActiveProject)-1 then
inc(i);
end;
finally
list.free;
end;

end;
```

Die Prozedur „ProofTimestamp“ (siehe Quelltext auf aktueller Seite) vergleicht die Zeitstempel der angelegten NE-Dateien des gerade aktiven Projektes mit der aktuellen Systemzeit.

Zu Beginn dieser Prozedur wird die Systemzeit in Sekunden seit dem 1. Januar 1970 bestimmt. Anschließend wird geprüft, ob zu den angegebenen NE im Projekt auch Zustandsdateien im NE Ordner existieren. Alle Einträge, zu denen keine Datei gefunden werden kann, erhalten im „array of record“ im Datenfeld „Value“ den Eintrag: „False“. Existiert zu einem Eintrag eine Datei, so wird deren Zeitstempel ermittelt und mit der Systemzeit verglichen. Ist dass Ergebnis kleiner als 86400s, dann ist die NE-Datei innerhalb der letzten 24 Stunden entstanden und wird somit als „Aktuell“ eingestuft und im entsprechenden „array of record“ wird im Datenfeld „Value“ der Wert „True“ hinterlegt. Ist die Datei älter als 24 Stunden wird der Wert „False“ übergeben und zusätzlich wird der Status des entsprechenden Netzelementes im „array-of-record“ auf ‚NA’ gesetzt.

```
Procedure ProofTimestamp(ProjectDir,NeDir,ActiveProject:string);
var
aktFileZeitForm : string;
entry : TSearchRec;
SystemDateInSeconds : longword;
FileDateInSeconds : longword;
i : integer;
aktFileZeit : TDateTime;
begin
SystemDateInSeconds:=DatumInSeconds(DOSDateAndTime);
For i:=0 to Listenlaenge(ProjectDir,ActiveProject)-1 do
begin
Ne[i].Value:=false;
If FuFileExists(NeDir+Ne[i].Name) then
if (Ne[i].Value = true) then
begin
FindFirst(NeDir+Ne[i].Name ,faAnyFile,entry);
aktFileZeit:=FileDateToDateTime(entry.Time);
aktFileZeitForm:=FormatDateTime ('yyyy.mm.dd_hh:mm:ss',aktFileZeit);
FileDateInSeconds:=DatumInSeconds(aktFileZeitForm);
Ne[i].TimeOK := SystemDateInSeconds - FileDateInSeconds;
if (Ne[i].TimeOK) < maxTimeOK then
begin
Ne[i].Value:=true;
end else
if (Ne[i].TimeOK) > maxTimeOK then
begin
Ne[i].Status:='NA';
Ne[i].value:= false;
end else if Ne[i].Name <>'' then
begin
Ne[i].value:= false;
end;
end;
end;

end;
```

Alle „aktuellen“ NE-Dateien des „aktiven“ Projektes werden in der Prozedur „NeDataCollect“ (siehe Quelltext auf aktueller Seite) innerhalb einer „For-Schleife“ ausgewertet. Die Funktion „CollectFromDataFile“ öffnet dabei sequentiell alle NE-Dateien des aktiven Projektes und durchsucht jede Datei nach den Schlüsselwörtern [STATUS] und [/STATUS] und ermittelt den dazwischen stehenden Wert. Dieser Wert wird dann an das jeweilige Statusfeld im „array-of-record“ des NE übergeben.

```
procedure NeDataCollect(ProjectDir,NeDir,ActiveProject:string);
var
FileToRead:text;
i:integer;
begin
For i:=0 to Listenlaenge(ProjectDir,ActiveProject)-1 do
begin
if (Ne[i].Value = true)then
begin
OpenTextForRead(NeDir+Ne[i].Name,FileToRead,1024);
Ne[i].Status := CollectFromDataFile(FileToRead,'[STATUS]','[/STATUS]',8,9);
CloseText(FileToRead);
OpenTextForRead(NeDir+Ne[i].Name,FileToRead,1024);
Ne[i].Host := CollectFromDataFile(FileToRead,'[HOST]','[/HOST]',6,7);
CloseText(FileToRead);
OpenTextForRead(NeDir+Ne[i].Name,FileToRead,1024);
Ne[i].LogFile := CollectFromDataFile(FileToRead,'[LOGFILE]','[/LOGFILE]',9,10);
CloseText(FileToRead);
OpenTextForRead(NeDir+Ne[i].Name,FileToRead,1024);
Ne[i].Contact:= CollectFromDataFile(FileToRead,'[CONTACT]','[/CONTACT]',9,10);
CloseText(FileToRead); }
end else
Ne[i].Status:='NA';
end;

end;
```

6 HTML Seiten erstellen

Aus den gesammelten Daten der vorhergehenden Prozeduren werden nun Webseiten zu jedem Projekt erstellt. Diese Webseiten bestehen aus 3 Frames.

Der obere Frame zieht sich über die komplette Seite und ist für den Namen des Projektes reserviert. Außerdem werden hier Buttons für den Start definierter Testrufe sowie pro Testruf jeweils eine Ergebniszeile angelegt. Der linke Frame wird eine Tabelle mit allen Netzelementen des aktuellen Projektes sowie deren Zustände enthalten. Den restlichen, überwiegenden Teil der Web-Seite ist für das Projektübersichtsbild vorgesehen.

Um die Frames mit den vorgesehenen Daten zu füllen, werden folgende Unterprozeduren verwendet:

„IndexHTML“

Die Prozedur „IndexHTML“ erstellt die „index.htm“ für das derzeit aktive Projekt. In dieser Datei wird die Einteilung der Web-Seite in die drei Frames definiert.

„BildHTML“

Über den Projektnamen des zurzeit aktiven Projektes in der „While-Schleife“ wird der Ordner ermittelt, indem sich das zugehörige Projektbild befindet. Dieses Bild wird aus diesem Ordner in den HTML-Ordner des Projektes kopiert.

Anschließend wird eine HTML-Datei mit dem Namen „bild.htm“ erzeugt, die als Inhalt dieses Projektübersichtsbild besitzt.

„TitelHTML“

Der Projektname des aktiven Projektes wird hier als Überschrift eingefügt. Außerdem findet sich in der „titel.htm“ die Bezeichnung des Testrufes, der das gesamte System testen soll. Nach der Ausführung des Test-Calls wird diese Seite aktualisiert und der Zustand des Systems wird farblich angezeigt.

„StatusHTML“

In der „status.htm“-Datei werden alle Namen und zugehörigen Zustandsinformationen der Netzelemente in einer Tabelle alphabetisch aufgelistet. Außerdem werden bestimmte Zustände farblich hinterlegt:

OK	„00FF00“	„grün“
MINOR	„FFFF00“	„gelb“
MAJOR	„FF9900“	„orange“
CRITICAL	„FF0000“	„rot“
NA	„999999“	„grau“

„ProjectOverview“

Hier wird eine Tabelle erzeugt/aktualisiert, die alle aktuellen Projekte enthält. Jeder Projektname in dieser Tabelle ist verlinkt mit der entsprechenden Index-Seite und kann so per Mausklick aufgerufen werden. Diese Übersicht dient als Startseite für den Benutzer. Durch Auswahl per Mausklick kommt man zum gewünschten Projekt. Die folgend dargestellte Prozedur („MakeHtmlFiles“) realsiert die gerade erläuterte HTML-Seitenerstellung.

```
procedure MakeHtmlFiles(ActiveProject:string);
begin
ActiveProject:=ExtractFileNameWithoutExt(ActiveProject);
if not DirectoryExists(IniData.Html+ActiveProject)then
MkDir(IniData.Html+ActiveProject); //HTML Seiten erstellen
IndexHTML(IniData.Html,ActiveProject);
StatusHTML(IniData.Html,IniData.Project,IniData.Ne,ActiveProject);
TitelHTML(IniData.Html,ActiveProject);
BildHTML(IniData.Html,IniData.XML,ActiveProject);
ProjectOverview(IniData.WebProgDir,IniData.Project,IniData.CGI_BIN);

end;
```

7 CFG-Files erstellen

Des Weiteren wird zu jedem Projekt ein CFG-File erstellt, welches später für die jeweiligen Testrufe benötigt wird. Hierbei sind 2 Unterfunktionen und 2 Prozeduren von größerer Bedeutung:

- Function FindTestSystemNum
- Function FindIpsqDir
- Procedure CFToCFGFile
- GenerateCFG

Zuallererst wird die Testanlage eines Projektes, mithilfe der Testanlagennummer, ermittelt, auf der die Testrufe ausgeführt werden. Es kommt häufiger auch vor, dass auf einer Testanlage mehrere Projekte laufen, allerdings niemals gleichzeitig, sondern immer zeitlich hintereinander.

Für die Ermittlung der Testanlagennummer wird die Funktion „FindTestSystemNum“ verwendet. Sie durchsucht das XML-File des aktiven Projektes nach dem Schlüsselwort „TA.NE::“ und gibt als Rückgabewert die ermittelte Zahl an die Funktion „FindIpsqDir“. In der Funktion „FindIpsqDir“ werden alle Unterverzeichnisnamen des Verzeichnisses der aktuell verwendeten Projekte („G:\html-testbed\p_act\“) mit der ermittelten Testanlagennummer verglichen und bei Übereinstimmung eine Zählvariable inkrementiert. Ist nach dem Vergleich mit alle Unterverzeichnissen die Zählvariable = „1“, so wird der Name des entsprechenden Verzeichnisses als Rückgabewert übergeben.

Wie bereits im vorletzten Absatz erwähnt, gibt es Projekte die über dieselbe Testanlage laufen. Es reicht also nicht, nur die Testanlagennummer zu wissen, sondern es muss eine weitere Differenzierung erfolgen. Hierzu wird die aktuelle „INI-Datei“ des Programmes „QuickTest“ [50] geöffnet und nach dem aktuell aktiven Projekt durchsucht. In dieser Datei stehen alle Testanlagen auf denen mehrere Projekte abwechselnd aktiv sind. Alle inaktiven Projekte werden mit „/*“ markiert. Die QuickT.ini wird zeilenweise nach der Testanlagennummer durchsucht. Jede Zeile, die die Testanlagennummer enthält, wird jeweils nach dem String „/*“ durchsucht. Aus der Zeile ohne „/*“ wird der Verzeichnispfad ausgeschnitten und als Rückgabewert übergeben.

[50] QuickTest ist ein NSN-internes Programm welches vorgefertigte Testrufe über eine grafische Benutzeroberfläche schnell und einfach manipulierbar macht. Der Tester ist somit in der Lage durch manuelle Änderung und ohne größeren Zeitaufwand Testrufe auf verschiedenen Testanlagen und Projekten auszuführen.

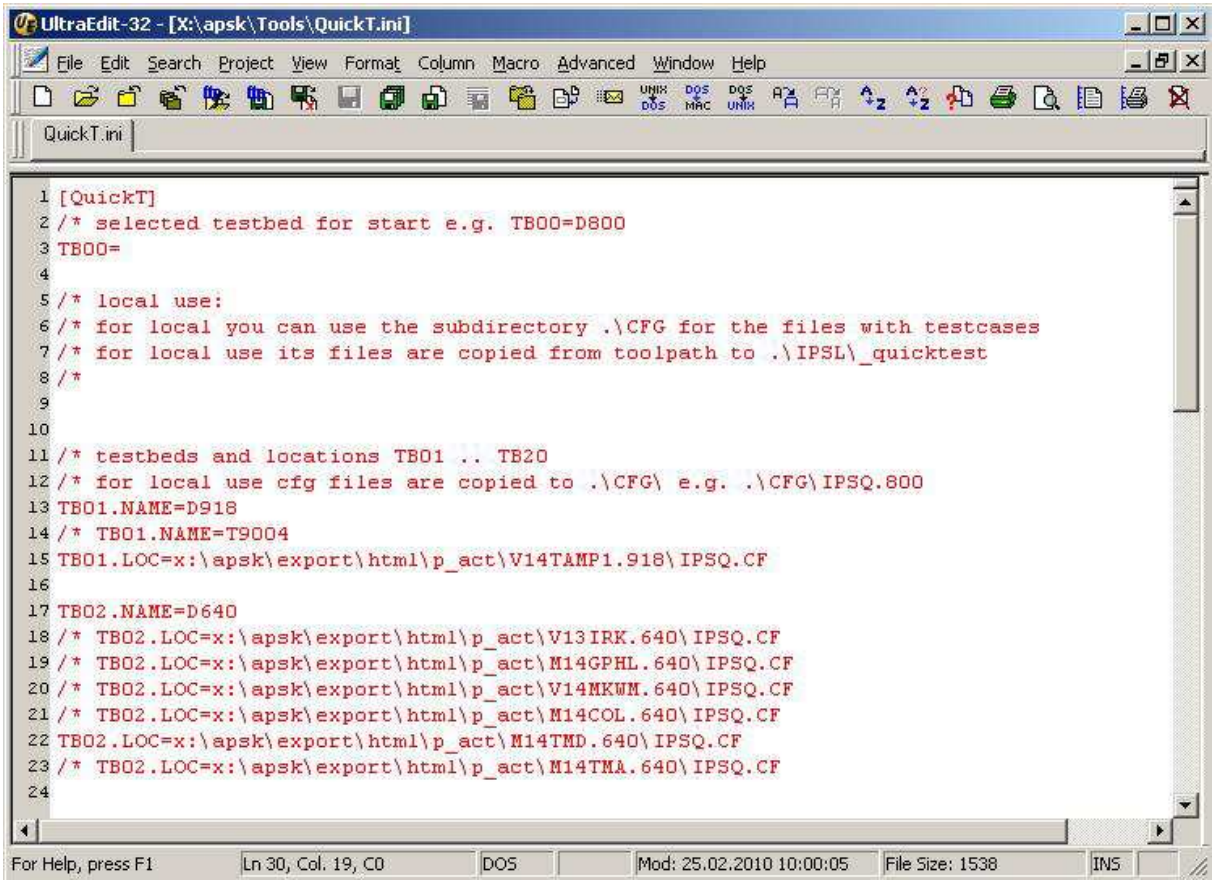

Abbildung 21: QuickT.ini

Mithilfe dieser beiden Funktionen ist der genaue Ordner der Testanlage des aktiven Projektes bestimmbar. In jedem Testanlagenordner befindet sich das Command-File(„ipsq.cf“), welches von „QuickTest“ als Quelldatei genutzt wird. Inhalt dieser Files sind vordefinierte Testrufszenarien zu den jeweiligen Projekten. Die Prozedur „CFToCFGFile“ kopiert einen definierten Testfall aus dem Command-File, speichert es als CFG-File und ergänzt folgende Informationen:

- Pfadangabe „CTL-File“
    - Pfad zum Control-File (“x:\Tools\ipsl_quicktest\bssap\cases\mmc\mmc”) des Mobile-Mobile-Call-Testszenarium, das in der vorliegenden Bachelorarbeit verwendet wurde
    - Wird ein anderes Testszenarium verwendet, muss auch ein anderes Control-File als Quelle angegeben werden.
- Speicherpfad für generiertes TRC-File
    - Zu jedem Projekt wird ein TRC-File erzeugt. Dieses File wird in den jeweiligen Html-Projektordner abgelegt.

- Call Duration / IMSI
    - Die Call Duration und die IMSI zu den beiden Testteilnehmern werden aus der „Main.ini“ in das CFG-File übertragen.
- Run-Mode
    - Als Run-Mode wird standardmäßig der Wert „2“ übergeben. Er bewirkt, dass nach der Durchführung eines Testrufes das jeweilige Fenster geschlossen wird.

In der Prozedur „GenerateCFG“ (siehe nachfolgenden Quelltext) werden nun die Funktion „FindIpsqDir“ und damit auch „FindSystemNum“ sowie die Prozedur „CFToCFGFile“ verwendet und mit den zuvor beschriebenen Variablen gefüllt. Die Parameter „CALL04“ und “END CALL04“ geben dabei den Anfang und das Ende der Beschreibung des MMC-Testcallszenarios in der „ipsq.cf“ des entsprechenden Projektes an.

```
procedure GenerateCFG(VisioHtmlDir,HtmlDir,IpsqDir,ActiveProject:string);
var
ProjectDir:string;
CFFile,CFGFile:string;
read,write:text;
begin
ProjectDir:=FindIpsqDir(IniData.QuickTIni,VisioHtmlDir,IpsqDir,ActiveProject);
if ProjectDir <> '' then
begin
ActiveProject:=ExtractFileNameWithoutExt(ActiveProject);
if FileExists(IPSQDir+ProjectDir+'\ipsq.cf') then
begin
CFFile:=IPSQDir+ProjectDir+'\ipsq.cf';
assign(read,CFFile);
CFGFile:=HtmlDir+ActiveProject+'\'+ActiveProject+'.cfg';
assign(write,CFGFile);
end;
If FileExists(HtmlDir+ActiveProject+'\'+ActiveProject+'.cfg') then
DeleteFile(HtmlDir+ActiveProject+'\'+ActiveProject+'.cfg');
OpenTextForRead(CFFile,Read,1024);
MyOpenTextForWrite(write,CFGFile);
CFToCFGFile(Read,Write,ActiveProject,'# CALL04 #','# END CALL04 #',0,0);
CloseText(Read);
MyCloseText(write);
end;

end;
```

Nachdem nun auch die Testcallszenarien für das aktuell geöffnete Projekt erstellt wurde, beginnt die „While-Schleife“ ab Punkt 4 (siehe Abbildung 19) mit dem nächsten Projekt erneut, bis alle Projekte aus dem Projektverzeichnis abgearbeitet sind. Unterschiedliche Zugriffsrechte im firmeninternen Netzwerk erfordern anschließend eine Synchronisation der Daten mit einem zusätzlichen Server.

8 Synchronisation

Innerhalb der Arbeitsgruppe wird regelmäßig eine Datensynchronisation durchgeführt. Da sich auf einem der Laufwerke außerdem der Zugangspunkt zum firmeninternen Web-Server befindet, werden die gesammelten Daten des „ENC“ und „TCA“ dorthin synchronisiert. Realisiert wird dies durch den Aufruf einer Kommando-Datei („sync.cmd“) und der Prozedur „ExecuteIt“.

```
ExecuteIt(IniData.SYNCBAT,false,true,true,true);
```

5.5.3 Entwicklung TCA

Neben der statischen Analyse der Netzelemente eines Projektes sollte, wie schon in der Vorüberlegung und im Lösungsansatz beschrieben, ein generischer Teil entwickelt werden, der es ermöglicht „On Demand“ die Kommunikation zwischen den Netzelementen zu testen. Eine Möglichkeit dies zu tun, ist die Ausführung und Analyse von sogenannten Testrufen. Der TCA - Test Call Analyzer sollte diese Aufgabe in Verbindung mit dem IPS-Tool übernehmen. Geplant war es, über einen Button auf den jeweiligen Web-Seiten der Projekte ein VB-Script ausführen zu lassen, welches den TCA startet. Im TCA wird dann der Aufruf des IPS-Tools mit den entsprechenden Informationen des gewählten Projektes geöffnet. Das IPS-Tool erstellt ein Auswertungsfile mit statistischen Informationen zum ausgeführten Szenario. Anschließend wertet der TCA diese Datei aus und aktualisiert die Web-Seite des aktuellen Projektes mit der Zusatzinformation zum Ergebnis des Testcalls.

5.5.4 Programmablauf TestCallAnalyzer (TCA)

In folgender Abbildung 22 wird der Programmablaufplan des TCA mit einer kurzen Beschreibung dargestellt. Eine nähere Erläuterung der einzelnen Abschnitte erfolgt im Anschluß.

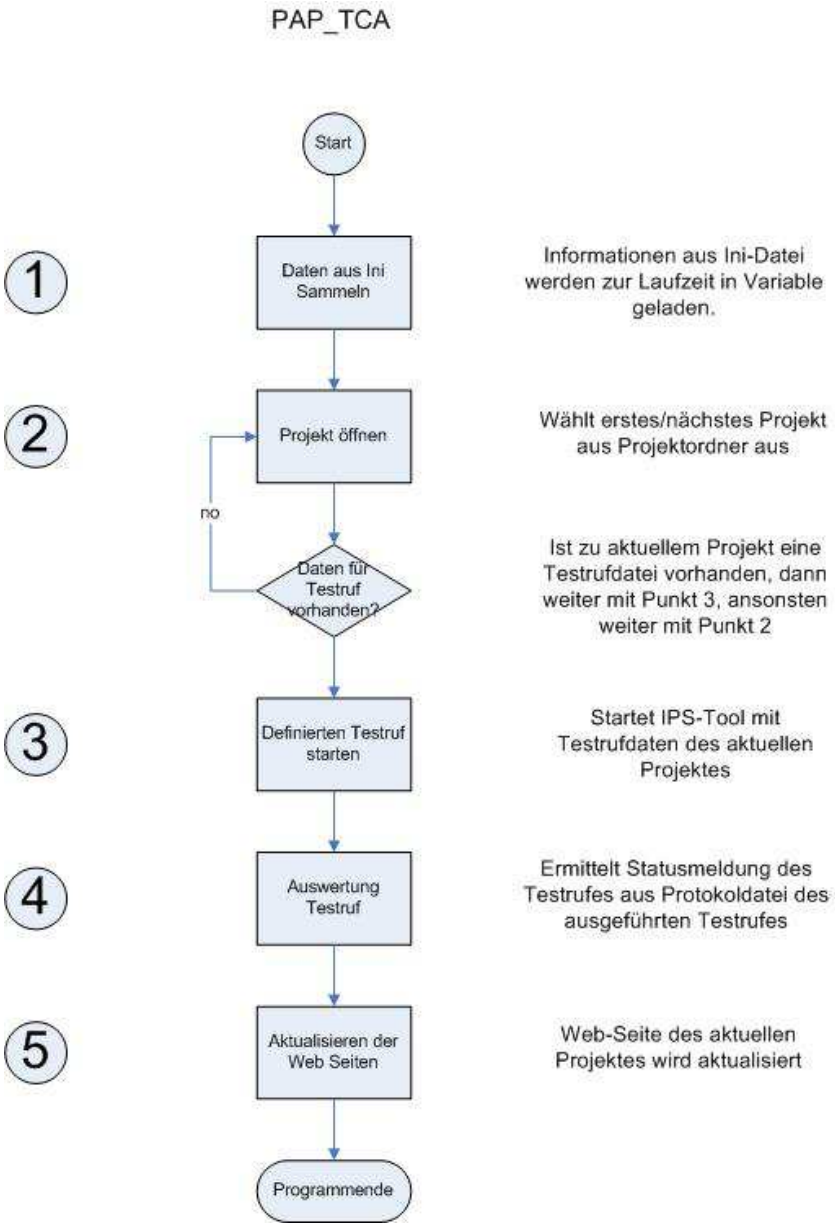

Abbildung 22: PAP_TCA

Die grundlegenden Prozeduren zum Öffnen, Schreiben und Schließen von Dateien „MyOpenTextForWrite“ und „MyCloseText“ sind bereits aus dem ENC bekannt und finden im TCA ebenfalls Verwendung.

Der TCA beginnt, genau wie der ENC, mit dem Einlesen von Initialisierungsdaten mithilfe der Prozedur „MyIniDataCollection“.

Für jedes Projekt werden nun folgende Schritte in einer „While-Schleife“ realisiert:

- Existenz eines gültigen CFG-Files im aktiven Projekt überprüfen
- Ausführen des IPS-Tools
- Auswertung des Ergebnisses des Testrufes

Eine korrekte Ausführung eines Testrufes zur Analyse eines Projektes kann nur durchgeführt werden, wenn das entsprechende CFG-File mit den korrekten Daten vorliegt. Mit der Funktion „FuFileExists“ wird getestet, ob im HTML-Ordner des aktiven Projektes ein CFG-File erstellt wurde. Es kommt vor, dass ein CFG-File zwar angelegt, aber kein Inhalt übergeben wird. Um diese Fehlerquelle zu ermitteln, wird die Funktion „GetFileSize“ verwendet. Sie ermittelt die Größe der CFG-Datei.

Mithilfe der Prozedur „ExecuteIt“ wird nun das IPS Tool gestartet und das CFG-File des aktiven Projektes ausgeführt.

Das IPS-Tool erstellt ein TRC-File, indem die Ergebnisse des Testrufes in definierter Form aufgelistet sind. Diese Statistik wird nun mithilfe der Prozedur „TestCallAnalyzing“ ausgewertet und die Ergebnisse in die Web-Seite des Projektes integriert. Für diese Auswertung sind die Funktion „CallOK“ und die Prozedur „CaptionHTML“ in die übergeordnete Prozedur „TestCallAnalyzing“ eingebunden.

Das vorliegende TRC-File wird nach dem Schlüsselwort: „Number of Not Failed Calls“ durchsucht. Der Wert hinter dieser Zeichenkette wird ausgewertet. Befindet sich eine Null hinter dem Schlüsselwort, dann gilt der Ruf als fehlerhaft. Bei jeder anderen Ziffer gilt der Testcall als erfolgreich. Die Prozedur aktualisiert nun die erstellte Web-Seite des Projektes und fügt die Auswertung des Testcalls visuell hinzu. Positive Testcalls werden dabei grün markiert und negative rot.

Sind alle Projekte abgearbeitet wird der TCA beendet.

5.6 Aufruf ENC und TCA

Beide Tools werden über Command-Files (siehe Abbildung 23) ausgeführt, die im Windows-Scheduler in festgelegten Zeitabständen gestartet werden. Sie sind also nicht durchgängig aktiv und eventuelle Fehler bei der Ausführung wirken sich somit nur temporär aus und führen nicht zwangsläufig zu einem dauerhaften Ausfall. Außerdem wird durch das Kommando „/min“ eine Möglichkeit geschaffen die beiden Programme minimiert zu starten. Der ENC wird alle 15 Minuten gestartet und der TCA alle 10 Minuten.

Abbildung 23: Aufruf ENC/TCA

5.7 Ergebnis

Die folgende Abbildung 24 zeigt abschließend beispielhaft die Web-Seite eines komplexen heterogenen Testsystems nach der Ausführung der beiden entwickelten Programme EasyNetChecker und TestCallAnalyzer.

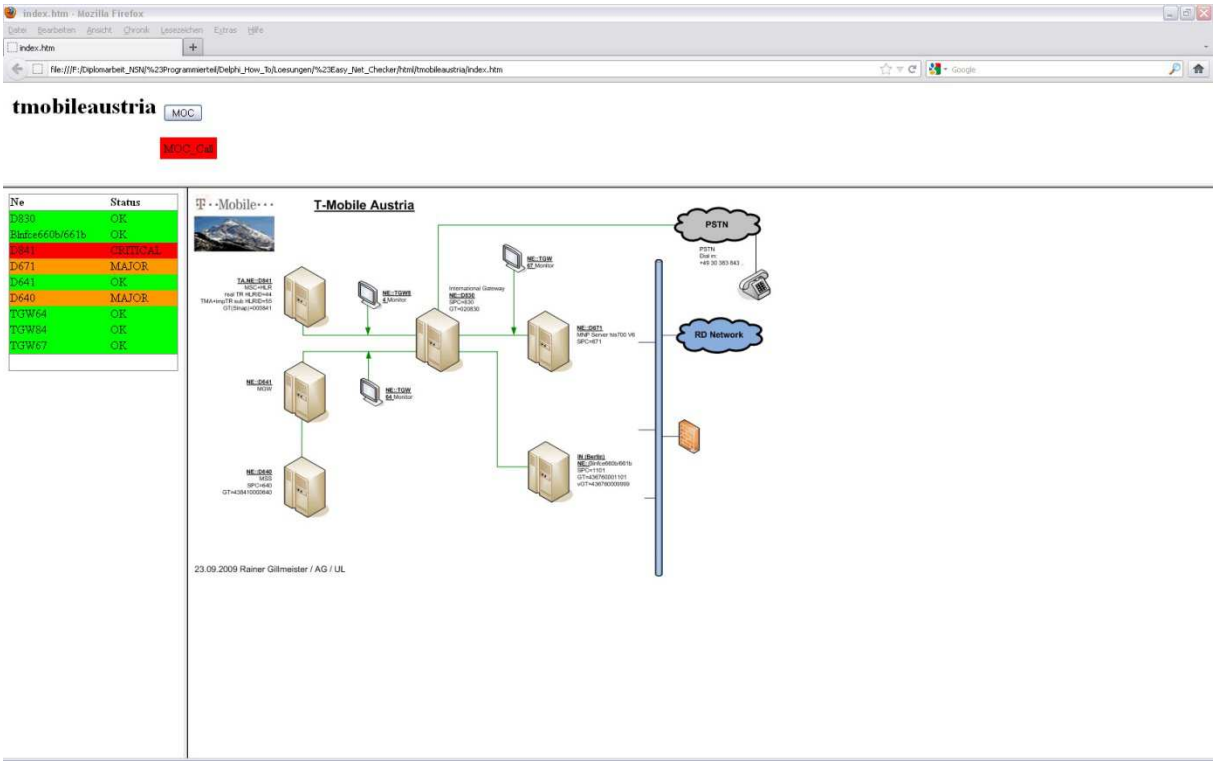

Abbildung 24: Beispiel Web-Seite

Im oberen Frame steht der Name des Testsystems. Er setzt sich aus dem Auftraggeber und dem Einsatzgebiet zusammen. Nebenstehend findet man einen Button, der für die Auslösung des entsprechenden Testcalls vorgesehen ist. Die Ausführung des Testcalls wird allerdings im vorliegenden Programm noch statisch ausgeführt und nicht dynamisch mit Mausklick auf den Button. Näheres dazu folgt in Kapitel 6. Direkt unter dem Button befindet sich die Statusanzeige des entsprechenden Testcalls. Diese gibt, mithilfe farblicher Unterlegung, das Ergebnis der Testcallauswertung wieder. Der rote Hintergrund vermittelt in diesem Falle, dass der Testcall nicht erfolgreich durchgeführt werden konnte.

Der große Frame unterhalb zeigt das Übersichtsbild des Testsystems mit allen beteiligten Netzelementen und deren Verbindungen zueinander.

Die Namen und die jeweiligen Zustände aller beteiligten Netzelemente findet man tabellarisch angeordnet links nebenstehend im dritten und letzten Frame der Web-Seite.

6 Zusammenfassung & Ausblick

Aufgabe der vorliegenden Bachelorarbeit war es, eine einfache Übersicht und Zustandsanzeige komplexer Mobilfunktestsysteme zu entwickeln. Die Mitarbeiter der Arbeitsgruppe sollten mithilfe dieser Anwendung in der Lage sein, die grundlegende Funktionalität aller Netzelemente eines Testsystems auf einen Blick zu erfassen. Außerdem sollte ermöglicht werden, auf Wunsch einen Testruf durchzuführen, um die Kommunikation zwischen den Netzelementen überprüfen zu können. Ziel war es, Fehler die auf Kommunikationsprobleme bzw. Netzelementeausfälle zurückzuführen sind, vor dem Ausführen komplexer Testszenarien zu erkennen. Verfälschte Testergebnisse in komplexen Tests sollten damit vermieden werden.

Der einleitende Teil dieser Arbeit beschäftigte sich mit der Analyse des Umfeldes in dem die Programme eingesetzt werden sollten, sowie der Analyse der gestellten Aufgabe. Dabei wurden Vorteile und Nutzen dieser Bachelorarbeit hervorgehoben. Des Weiteren wurde die Einordnung der entwickelten Anwendung in das Umfeld untersucht und es erfolgte eine Einarbeitung in die zu verwendende Programmiersprache Delphi. Die Planung und Gliederung der weiteren Arbeitsschritte vervollständigte den einleitenden Teil dieser Bachelorarbeit.

Die zu entwickelnde Software soll von einem Mitarbeiter gepflegt und weiterentwickelt werden. Deshalb war eine gute Dokumentation erforderlich. Außerdem sollte möglichst Software verwendet werden, die bereits in der Arbeitsgruppe vorhanden war. Bei der Entwicklung der Software war darauf zu achten, dass der Einsatz auf einem Desktop-PC eines Mitarbeiters im Hintergrund laufen sollte. Die Software musste also möglichst ressourcensparend und als Hintergrundprogramm entwickelt werden. Die Entwicklung einer Software mit einem GUI bot sich an dieser Stelle deshalb nicht an. Stattdessen wurde sich für die Konsolenprogrammierung entschieden.

Zum Softwarerepertoire der Arbeitsgruppe gehört u. a. Delphi, welches als Entwicklungsumgebung genutzt werden sollte. Delphi unterstützt die Entwicklung von Konsolenanwendungen. Neben der Reduzierung des Ressourcenbedarfs wird in Delphi auch die Möglichkeit gegeben, ein Programm sofort minimiert im Hintergrund zu starten. Die Einarbeitung in diese Programmiersprache wurde durch die klaren Strukturen und einprägsamen Befehle erleichtert. Ein kontextbezogenes Hilfesystem sowie die Anzeige möglicher Methoden und Eigenschaften während der Quelltexterstellung waren ebenfalls sehr hilfreich. Außerdem bietet Delphi die Möglichkeit, eigene Komponenten zu entwickeln und diese dem Programm als Vorlage hinzuzufügen. Bereits vorhandene Softwaremodule konnten so einfach eingebunden und verwendet werden.

Während der weiterführenden Planung und Gliederung wurden Meilensteine in der Entwicklung der Software definiert und daraus auch die einzelnen Kapitel dieser Abschlussarbeit festgelegt. Die Einarbeitungsphase ist dabei der erste Meilenstein. Dazu gehörten die weiter oben beschriebene Einleitung mit der Aufgabenanalyse und die Einarbeitung in Delphi. Ebenfalls Teil dieses Meilensteines war es, die Grundlagen des GSM-Netzes und dabei das Arbeitsumfeld der Arbeitsgruppe kennen und verstehen zu lernen. Ein Ergebnis dieses Meilensteines war die Gliederung der Abschlussarbeit.

Der zweite Meilenstein beinhaltete die Analyse bereits entwickelter Software. Ziel war es dabei, die Notwendigkeit der hier vorgestellten Entwicklung erkennbar zu machen.

Die gewonnenen Erkenntnisse wurden zusammen mit den eigenen Ideen anschließend im dritten Meilenstein, in den Vorüberlegungen zur eigenen Softwareentwicklung, formuliert und ermöglichten im Anschluss die Definition eines fertigen Lösungsansatzes.

In der vorliegenden Bachelorarbeit ist eine Anwendung entwickelt worden, mit deren Hilfe man den Zustand komplexer Testsysteme (Testprojekte) und dessen Einzelelemente visualisieren kann. Dabei wurde gewährleistet, dass das Hinzufügen und Entfernen von Einzelelementen und ganzen Testsystemen, unter Einhaltung der beschriebenen Vorschriften, einfach realisierbar ist. Der entscheidende Punkt war die Definition einer allgemeinen Schnittstelle zwischen den verschiedensten Netzelementen und der entwickelten Anwendung. Nötige Tests zur Bestimmung der Netzelementezustände werden von den jeweiligen Betreuern regelmäßig und automatisch durchgeführt. Hier kommt ein einfaches Filesystem zur Anwendung. Die Ergebnisse der Zustandsermittlungen werden in beschriebener, fest definierter Form in einem Ordner als Textdatei abgelegt. Diese NE-Dateien bilden zusammen mit den beschriebenen anderen Projektdateien die Grundlage für die entwickelte Lösung.

Die Anwendung besteht aus zwei Teilen, dem ENC – Easy Net Checker und dem TCA – Test Call Analyzer. Wichtige Variablen beider Programmteile sind in einer Initialisierungsdatei („Main.ini“) hinterlegt, um so dem Benutzer die Möglichkeit zu geben, ohne großes Verständnis des Quelltextes später eventuell nötige Anpassungen im Filesystem oder anderer wichtiger Ausgangsbedingungen vorzunehmen.

Der erste Programmteil, der Easy Net Checker, realisiert die Sammlung der Zustandsdaten aller aktuellen Netzelemente. Anschließend werden alle zugehörigen Netzelemente eines Projektes durch Auswerten der entsprechenden XML-Datei bestimmt. Der ENC arbeitet nun sequentiell alle NE-Dateien des Projektes ab und ermittelt die aktuellen Zustände aus den entsprechenden NE-Dateien. Zum Schluss erstellt der Easy Net Checker eine Web-Seite des aktuellen Projektes. Dabei werden die ermittelten Zustandsinformationen in Form einer

Tabelle zusammengestellt. Das entsprechende Übersichtsbild zum Projekt wird aus den Projektdateien, wie beschrieben, ermittelt und ergänzt die Web-Seite.

Der ENC wiederholt diese Vorgänge für alle im aktuellen Testsystemeordner hinterlegten Projekte.

Im zweiten Programmteil, dem Test Call Analyzer (kurz:"TCA"), werden die Tests von Projekten als Ganzes realisiert. Dabei werden definierte Testrufe ausgelöst, die das Zusammenspiel der NE der entsprechenden Projekte testen (mithilfe des IPS-Tools). Die vom „ENC“ angelegten Projektverzeichnisse werden sequentiell geöffnet und nach den entsprechenden Konfigurationsdateien (enthalten Beschreibung des auszuführenden Testrufes) durchsucht. Sind diese Dateien vorhanden, wird das „IPS-Tool“ ausgeführt und der entsprechende Ruf ausgeführt. Das „IPS-Tool“ erstellt während der Anwendung ein „TraceFile“, indem sich Statistiken zum durchgeführten Testruf befinden. Diese erstellten Files werden vom TCA gefiltert. Dabei wird ermittelt, ob der Testruf erfolgreich war oder nicht. Das Ergebnis wird zum Schluss in die Web-Seiten der Projekte integriert und visualisiert.

Im Laufe der Entwicklung gab es verschiedenste Hürden. So war es geplant, Testrufe von der jeweiligen Webseite eines Projektes über Buttons auszulösen und nach der Auswertung der Ergebnisse die Webseite zu aktualisieren. Dabei entstand die Idee über ein VB-Skript, wie in Abbildung 25 dargestellt, ein zweites Programm zu starten mit dem Testrufe ausgelöst und ausgewertet werden können und welches anschließend die Webseiten aktualisiert und dabei die Ergebnisse der Testrufe visualisiert.

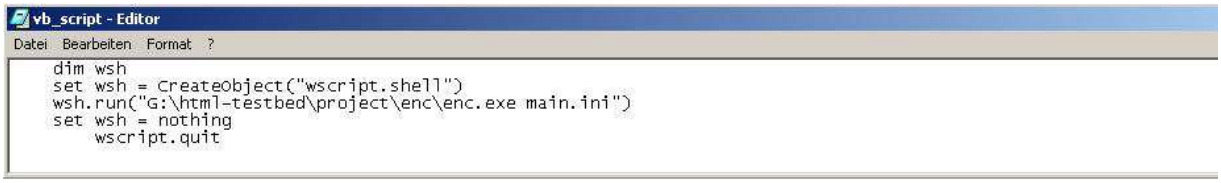

Abbildung 25: Beispiel_VB_Skript

Die Verwendung jeglicher Skripte, die ausführbare Programme aufrufen können (notwendig für den Aufruf des IPS-Tools), wird von der innerbetrieblichen Web-Server-Applikation, aus Sicherheitsgründen, nicht gestattet. Die Realisierung einer dynamisch-startbaren Analyse über den Web-Browser ist somit nur schwer umzusetzen gewesen und konnte im Zeitrahmen nicht realisiert werden. Die Analyse von Gesamtprojekten wurde stattdessen vorläufig, wie die der NE, statisch realisiert (siehe „Aufruf ENC und TCA“). Mit Abstimmung des zuständigen Administrators ist im Verlaufe der Weiterentwicklung eine Realisierung der dynamisch-startbaren Analyse aber nicht ausgeschlossen.

Bei der Auswertung der Netzelementedateien wird im „ENC“ bisher nur die Zeile des Status ausgewertet. Die Zusatzinformationen in diesen Dateien könnten in Zukunft ergänzend auf den jeweiligen Web-Seiten der Projekte dauerhaft oder per Mausklick angezeigt werden. Per Klick auf ein NE eines Projektes würde man so nähere Informationen erhalten. Die Auswertung kann im „ENC“ einfach über die wiederholte Verwendung der Funktion „CollectFromDataFile“ durchgeführt werden (siehe Kapitel 5 – „NeDataCollect“). Dabei bleibt der Aufbau gleich und man ändert nur die entsprechenden Schlüsselwörter und deren Parameter. Der ermittelte Wert wird, genau wie der Status, an ein entsprechendes Feld im „Array-of-Record“ übergeben und kann nun wie der Status auf gewünschte Weise in die entsprechenden Web-Seiten integriert werden.

Während der Entwicklung des ENC wurde die Funktion des Einsammelns von NE-Zustandsdateien per FTP Testweise an einer Datei und an einem FTP-Server durchgeführt. Durch ergänzende Angaben der zu kopierenden Dateien und verwendeter FTP-Server und deren Zugriffsdaten ist auch hier eine Erweiterung des Leistungsumfanges des „ENC“ denkbar.

Im vorliegenden Programm wird ein sogenannter Mobile-Mobile-Call mithilfe des IPS-Tools initiiert und ausgewertet. An dieser Stelle ist es denkbar, den Leistungsumfang der entwickelten Lösung zu erhöhen, indem man weitere unterschiedliche Testcallszenarien wie zum Beispiel: SMS, GPRS, oder USSD integriert. Dazu muss man nur die nötigen IPS-Szenarien in das entwickelte Tool einbinden und anschliessend das Format der Web-Seite anpassen. Auf diese Weise können entsprechend zuständige Netzelemente wie z. B. der SGSN und GGSN auf deren Zusammenspiel getestet werden. Die entsprechenden Szenarien findet man, genau wie das „MMC-Szenario“, in der „ipsq.cf“ eines jeden Projektes. Die Einbindung der Testszenarios kann also genau wie unter Kapitel 5 – „CFG Files erstellen“ erfolgen.

Ein weiterer möglicher Ansatz der Weiterentwicklung wäre die Realisierung der Zustandsanzeigen direkt im Übersichtsbild, indem man die einzelnen Netzelemente je nach Zustand farblich hinterlegt.

Der entwickelte Easy Net Checker betrachtet in jedem Projekt eine definierte Testanlage. Es existieren allerdings auch Projekte in denen mehrere Testanlagen aktiv sind. Um hier die entwickelte Lösung flexibler zu Gestalten, wäre eine sinnvolle Erweiterung des vorliegenden Tools eine Methode der Selektion dieser Testanlagen zu ermöglichen.

Die Verwendung dieser Lösung wird den Mitarbeitern durch die frühzeitige Erkennung elementarer Fehler Zeitaufwand ersparen und auch im Hinblick auf die möglichen Weiterentwicklungen die Qualität der gesamten Arbeitsgruppe weiter anheben.

Literaturverzeichnis

(Alekseev 2010)	Alekseev, Olga. *Konzeption und Entwicklung einer Software zur Visualisierung von Dienstabläufen Intelligenter Netze bei der Siemens AG.* Berlin, 2010.
(Delphi Treff: Fehlerbehandlung 2010)	Delphi Treff: Fehlerbehandlung. *Delphi Treff.* 2010. http://www.delphi-treff.de/tutorials/ (Zugriff am 10. Januar 2010).
(Elektronik Kompendium 2010)	Elektronik Kompendium. *Elektronik Kompendium : Geschichte Mobilfunk.* 2010. http://www.elektronik-kompendium.de/sites/kom/0910121.htm (Zugriff am 22. Februar 2010).
(Heise MRTG 2010)	Heise MRTG. *Heise online.* 2010. http://www.heise.de/software/download/multi_router_traffic_grapher/5800a (Zugriff am 01. März 2010).
(Heise MRTG_Bild 2010)	Heise MRTG_Bild. *Heise Online.* 2010. http://www.heise.de/download/multi-router-traffic-grapher-12015800.html (Zugriff am 01. März 2010).
(Heise: Xymon 2010)	Heise: Xymon. *Heise online.* 2010. http://www.heise.de/software/download/xymon/63385.html (Zugriff am 02. März 2010).
(ITWissen_Lexikon : IN 2010)	ITWissen_Lexikon : IN. *"IT-Wissen - IT-Lexikon für Internet, Telekommunikation, Software und Elektronik".* 2010. http://www.itwissen.info/definition/lexikon/Intelligentes-Netz-IN-intelligent-network.html (Zugriff am 22. Februar 2010).
(ITWissen_Lexikon: NGMN 2010)	ITWissen_Lexikon: NGMN. *"IT-Wissen - IT-Lexikon für Internet, Telekommunikation, Software und Elektronik".* 22. Februar 2010. http://www.itwissen.info/definition/lexikon/next-generation-mobile-network-NGMN.html (Zugriff am 22. Februar 2010).

(LFM-NRW: GPRS 2010) LFM-NRW: GPRS. *LTE-Projekt NRW.* 2010. http://www.lte-projekt-nrw.de/teedrei/Vorlaeufer.146.0.html (Zugriff am 23. Februar 2010).

(MobileIN.com 2010) MobileIN.com. *Welcome to MobileIN.com - Wireless and Mobile Communications.* 2010. www.mobilein.com/Mobile%20Intelligent%20Networking.pdf (Zugriff am Februar 2010).

(Mobilfunk 2010) Mobilfunk, ITWissen_Lexikon:. *"IT-Wissen - IT-Lexikon für Internet, Telekommunikation, Software und Elektronik".* 2010. http://www.itwissen.info/definition/lexikon/Mobilfunk-cellular-radio.html (Zugriff am 22. Februar 2010).

(Nokia Siemens Networks 2010) Nokia Siemens Networks. „NSN Intranet." 2010. https://sharenet-ims.inside.nokiasiemensnetworks.com/livelink/livelink/400803682 (Zugriff am 10. Februar 2010).

(Nokia Siemens Networks 2010) Nokia Siemens Networks. „NSN Intranet." 2010. https://sharenet-ims.inside.nokiasiemensnetworks.com/livelink/livelink (Zugriff am 10. Februar 2010).

(Peter 2010) Peter, Anka. *Entwurf und Realisierung von CAMEL 2 Interoperabilitätstests.* Berlin, 2010.

(Quest Software: BigBrother 2010) Quest Software: BigBrother. *Quest Software GmbH.* 2010. http://www.questsoftware.de/bigbrother/features-benefits.aspx (Zugriff am 21. Februar 2010).

(Riemer, R. 2010) Riemer, R. *UMTSLink: GPRS_GGSN.* 2010. http://www.umtslink.at/index.php?pageid=GPRS_ggsn (Zugriff am 23. Februar 2010).

(Riemer, R. 2010) Riemer, R. *UMTSLink: GPRS_SGSN.* Februar 2010. http://www.umtslink.at/index.php?pageid=GPRS_sgsn (Zugriff am 23. Februar 2010).

(Riemer, R. 2010)	Riemer, R. *UMTSLink: UTRAN.* 2010. http://www.umtslink.at/index.php?pageid=utran (Zugriff am 23. Februar 2010).
(UMTSLink: Intelligent Network 2010)	UMTSLink: Intelligent Network. *www.UMTSlink.at.* 2010. http://www.umtslink.at/content/inarchitektur-83.html (Zugriff am 20. Februar 2010).
(Uni Zürich: Software engineering 2010)	Uni Zürich: Software engineering. „Universität ZürichDepartment of Informatics.“ Februar 2010. http://www.ifi.uzh.ch/rerg/fileadmin/downloads/teaching/courses/software_engineering_ws0506/Kapitel_08_Testen.pdf (Zugriff am Februar 2010).
(Wikipedia 2010)	Wikipedia. *Nagios.* 2010. http://de.wikipedia.org/w/index.php?title=Datei:Nagios_service_detail.png&filetimestamp=20081113222520 (Zugriff am 01. März 2010).
(Wikipedia, Nagios 2010)	Wikipedia. *Nagios.* 2010. http://de.wikipedia.org/wiki/Nagios (Zugriff am 01. März 2010).
(Wikipedia, Softwaretest 2010)	Wikipedia. *Softwaretest.* 2010. http://de.wikipedia.org/wiki/Softwaretest (Zugriff am 02. März 2010).
(Wikipedia: GSM 2010)	Wikipedia: GSM. *Wikipedia.* 2010. http://de.wikipedia.org/wiki/Global_System_for_Mobile_Communications (Zugriff am 22. März 2010).
(Wikipedia: NGMN 2010)	Wikipedia: NGMN. 2010. http://de.wikipedia.org/wiki/Next_Generation_Mobile_Networks (Zugriff am 23. Februar 2010).
(Xymon 2010)	Xymon. *Beispiel_Xymon.* 2010. http:://www.xymon.com/hobbit/rep/3828-1247297319/servers/servers.html (Zugriff am 01. Februar 2010).

Anlagen

Quellcode ENC

```
{$APPTYPE CONSOLE}
// Parameter 1 = Main.Ini
program enc;
//---------------------------------------------------------------------------
uses
classes,
IniFiles,
MyExceptH,
Sysutils,
Diverses,
FilesU,
timecnvu,
FileCtrl,
ProcessU;
// TYPE,CASE, TSearchRec, TMemIniFile, FileExists, CreateIOException,
//---------------------------------------------------------------------------
type
TNetelement = record //Record Definition
Name : string; //
Status : string; //
Host : string; //
LogFile : string; //
Contact : string; //
TimeOK : longword; //
Value : boolean; // -----------------
end;
Type TIniEntry = record
Ne : string;
Project : string;
Html : string;
XML : string;
CFGFile : string;
HTML_G : string;
ProgDir : string;
IPSQDir : string;
CGI_BIN : string;
WebProgDir : string;
QuickTIni : string;
SYNCBAT : string;
NEDATAFTP : string;
ENCCMD : string;
TCACMD : string;
end;
Type TTlnData = record
DUR_A : string;
DUR_B : string;
IMSI_A : string;
IMSI_B : string;
end;
const MaxNe = 2000; //legt die maximale Anzahl der verwaltbaren NE's fest
maxTimeOK = 86400; //24*60*60 --> sekunden eines Tages --> bei überschreitung,
//datei zu alt
var
Failure,debug:boolean;
Ne : array [0..MaxNe] of TNetelement; //Liste der NE
IniData:TIniEntry;
TlnData:TTlnData;
```

```
FileSearch: integer;
entry:TSearchRec;
Project:string;
//---------------------------------------------------------------------------
procedure MyIniDataCollection; //Auslesen aller wichtigen Daten aus der Ini
var
ini : TMemIniFile;
begin
ini:=TMemIniFile.create(ParamStr(1));
try
//Verzeichnis des NeOrdners an Var übergeben
IniData.Ne := ini.ReadString('SOURCE','NE','');
//Verzeichnis der Visiohtml Seiten
IniData.XML := ini.ReadString('SOURCE','XML','');
IniData.IPSQDir := ini.ReadString('SOURCE','IPSQDIR','');
IniData.SYNCBAT := ini.ReadString('SOURCE','SYNCBAT','');
IniData.NEDATAFTP := ini.ReadString('SOURCE','NEDATAFTP','');
IniData.ENCCMD := ini.ReadString('SOURCE','ENCCMD','');
IniData.TCACMD := ini.ReadString('SOURCE','TCACMD','');
IniData.WebProgDir := ini.ReadString('SOURCE','WEBPROGDIR','');
IniData.QuickTIni := ini.ReadString('SOURCE','QUICKTINI','');
IniData.Project := ini.ReadString('DESTINATION','PROJECT',''); //Verzeichnis des ProjectOr
IniData.Html := ini.ReadString('DESTINATION','HTML',''); //Verzeichnis der generiert
//IniData.CFGFile := ini.ReadString('DESTINATION','CFG',''); //CFG Files zu Projecten
IniData.HTML_G := ini.ReadString('DESTINATION','HTML_G','');//
IniData.ProgDir := ini.ReadString('DESTINATION','PROGDIR','');
IniData.CGI_BIN := ini.ReadString('DESTINATION','CGIDIR','');
TlnData.DUR_A := ini.ReadString('IPSQ','DUR_A','');
TlnData.DUR_B := ini.ReadString('IPSQ','DUR_B','');
TlnData.IMSI_A := ini.ReadString('IPSQ','IMSI_A','');
TlnData.IMSI_B := ini.ReadString('IPSQ','IMSI_B','');
finally
ini.free;
end;
end;
//---------------------------------------------------------------------------
procedure MyOpenTextForWrite(VAR g : Text;Name : string);
begin
if FileExists(Name) then begin
{$I-}
Assign(g, Name); {Assign --> zuordnen --> Variable "g" bekommt den Namen zugewiesen}
Append(g); {hinzufügen/anfügen von "g"}
CreateIOException(IOResult, 'Append', g, true); {vergleich ob datei schon vorhanden, wenn ja, f
{$I+}
end else begin
{$I-}
Assign(g, Name);
Rewrite(g);
CreateIOException(IOResult, 'Rewrite', g, true);
{$I+}
end;
end;
//---------------------------------------------------------------------------
procedure MyCloseText(VAR g : Text);
begin
//{$I-}
case TTextRec(g).Mode of
fmInput, fmOutput, fmInOut:
begin
Close(g);
CreateIOException(IOResult, 'Close', g, true);
end;
end
//{$I+}
```

```
end;
//---------------------------------------------------------------------
procedure CollectXmlData(var FileToRead,FileToWrite:text; StartKeyword,EndKeyword : string; StartKe
var
DataSearch :string;
position :Integer; //funktion XmlZeileNe ob zeilen in xml, Net
begin
while not eof(FileToRead) do
begin
readln(FileToRead,Datasearch);
position:= pos(StartKeyword,Datasearch);
if position <> 0 then
begin
DataSearch:=copy(DataSearch,position,20);
position:=pos(StartKeyword,DataSearch);
delete(Datasearch,position,StartKeyLength);
DataSearch:=MakePartStr(DataSearch,' ',1);
position:=pos(EndKeyword,DataSearch);
if position <> 0 then
delete(DataSearch,position,EndKeyLength);
position:= pos(' ',Datasearch);
if position <> 0 then
begin
DataSearch:=MakePartStr(DataSearch,' ',1); //Searchstring wird nach leerzeichen abgeschn
Datasearch:=Datasearch+'.txt';
writeln(FileToWrite,Datasearch);
end else begin
DataSearch:=trim(Datasearch);
DataSearch:=DataSearch+'.txt';
writeln(FileToWrite,DataSearch);
end;
end;
end;
end;
//---------------------------------------------------------------------
procedure ReadXmlFiles(VisioDir,ProjectDir:string);
var entry : TSearchRec;
ProjectName : string;
FileSearch : Integer;
FileToRead,FileToWrite : text;
begin
FileSearch:=FindFirst(VisioDir+'*.htm' ,faAnyFile,entry); //Pointer auf File im N
while FileSearch = 0 do
begin
ProjectName := ExtractFileNameWithoutExt(entry.name); //filename ohne .*
if FileExists(VisioDir+ProjectName+'_files\data.xml') then //Englische Visio Version
OpenTextForRead(VisioDir+ProjectName+'_files\data.xml',FileToRead,1024) //Pfadangabe + Textdatei
else
OpenTextForRead(VisioDir+ProjectName+'-Dateien\data.xml',FileToRead,1024); //Deutsche Visio Version
MyOpenTextForWrite(FileToWrite,ProjectDir+ProjectName+'.txt');
CollectXmlData(FileToRead,FileToWrite,'NE::','</Text>',4,7);
CloseText(FileToRead);
MyCloseText(FileToWrite);
FileSearch:=FindNext(entry);
end;
FindClose(entry);
end;
//---------------------------------------------------------------------
function ListLength(ProjectDir,ActiveProject:string):integer;
var
list:TStringList;
ProjectCountedNe:integer;
begin
list:=TStringList.Create; //immer nötig, bevor man listen verwenden kann
```

```
list.LoadFromFile(ProjectDir+ActiveProject); //lade liste des gewünschten Projectes
try
ProjectCountedNe:=list.Count;
Result:=ProjectCountedNe; //Rückgabe Anzahl Listeneinträge
finally
list.free;
end;
end;
//---------------------------------------------------------------------------
Procedure GetNeNames(ProjectDir,ActiveProject:string); // sammeln der Ne Namen eines Projectes
var
i,j : integer;
list:TStringList;
begin
list:=TStringList.Create; //immer nötig, bevor man listen verwenden ka
list.LoadFromFile(ProjectDir+ActiveProject); //lade liste des gewünschten Projectes
list.Sort;
i:=0;
try
for j:=0 to ListLength(ProjectDir,ActiveProject)-1 do
begin
Ne[i].Name:=list.Strings[j]; //wenn NeName in Liste des Projectes
if i<ListLength(ProjectDir,ActiveProject)-1 then
inc(i);
end;
finally
list.free;
end;
end;
//---------------------------------------------------------------------------
Procedure ProofTimestamp(ProjectDir,NeDir,ActiveProject:string); // vergleicht NeDateizeit mit S
var
aktFileZeitForm : string;
entry : TSearchRec;
SystemDateInSeconds : longword;
FileDateInSeconds : longword;
i : integer;
aktFileZeit : TDateTime;
begin
SystemDateInSeconds:=DatumInSeconds(DOSDateAndTime); //ermittelt aktuelle SystemZeit
For i:=0 to ListLength(ProjectDir,ActiveProject)-1 do
begin
Ne[i].Value:=false;
If FuFileExists(NeDir+Ne[i].Name) then // Existiert das NE im NeOrdner
Ne[i].Value:=true;
if (Ne[i].Value = true) then
begin
FindFirst(NeDir+Ne[i].Name ,faAnyFile,entry); //Pointer auf File im
aktFileZeit:=FileDateToDateTime(entry.Time); //ermittelt Datum und
aktFileZeitForm:=FormatDateTime ('yyyy.mm.dd_hh:mm:ss',aktFileZeit); //Format festlegen, n
FileDateInSeconds:=DatumInSeconds(aktFileZeitForm); //Datum und Zeit in s
Ne[i].TimeOK := SystemDateInSeconds - FileDateInSeconds; //ermikttelt Zeitdiff
if (Ne[i].TimeOK) < maxTimeOK then //Zeit des Ne kleiner maxTo
begin
Ne[i].Value:=true;
end else
if (Ne[i].TimeOK) > maxTimeOK then //wenn max Toleranzzeit überschritten dann...
begin
Ne[i].Status:='NA'; //status auf Failure gesetzt
Ne[i].value:= false; //value auf false --> keine überprüfung mehr
end else if Ne[i].Name <>'' then //wenn kein name in array eingetragen dann...
begin
Ne[i].value:= false; //value = false --> keine überprüfung mehr
end;
```

```
end;
end;
end;
//---------------------------------------------------------------------------
function CollectFromDataFile(var FileToRead:text; StartKeyword,EndKeyword : string; StartKeyLength,
var
DataSearch,DataSearch1 :string;
position :Integer;
begin
result:='';
while not eof(FileToRead) do
begin
readln(FileToRead,DataSearch); //zeile einlesen, übergabe an Datasearch
position:= pos(StartKeyWord,DataSearch); //setze position an konst. string
if position <> 0 then //wenn string vorhanden dann...
begin
delete(DataSearch,position,StartKeyLength); //löscht const String
position:= pos(EndKeyword,DataSearch); //setzt position an konst string
if position <> 0 then //wenn string vorhanden, dann...
begin
delete(DataSearch,position,EndKeyLength); //löscht const String
result:=DataSearch //übergabe text zwischen konstanten an Ne[i].*
end else
repeat
readln(FileToRead,DataSearch1); //wenn KeyEnde nicht in einer Zeile
DataSearch:=DataSearch+DataSearch1; //mit KeyAnfang, dann nächste zeile einlesen
position:= pos(EndKeyword,DataSearch); //und mit vorhergehender Zeile verbinden
until (position <> 0);
delete(DataSearch,position,EndKeyLength); //wenn ja dann löschen Keyende und ausgabe res
result:=trim(DataSearch);
end;
end;
end;
//---------------------------------------------------------------------------
procedure NeDataCollect(ProjectDir,NeDir,ActiveProject:string); // Sammelt Namen der Ne's, überg
var
FileToRead:text;
i:integer;
begin
For i:=0 to ListLength(ProjectDir,ActiveProject)-1 do
begin
if (Ne[i].Value = true)then
begin
OpenTextForRead(NeDir+Ne[i].Name,FileToRead,1024); //Öffnen Datei aus
Ne[i].Status := CollectFromDataFile(FileToRead,'[STATUS]','[/STATUS]',8,9); //Auslese
CloseText(FileToRead); //Schließen D
{ Für die Zukunft --> Erweiterungen der Statusdatei
OpenTextForRead(NeDir+Ne[i].Name,FileToRead,1024); //Öffnen Datei aus
Ne[i].Host := CollectFromDataFile(FileToRead,'[HOST]','[/HOST]',6,7);
CloseText(FileToRead);
OpenTextForRead(NeDir+Ne[i].Name,FileToRead,1024);
Ne[i].LogFile := CollectFromDataFile(FileToRead,'[LOGFILE]','[/LOGFILE]',9,10);
CloseText(FileToRead);
OpenTextForRead(NeDir+Ne[i].Name,FileToRead,1024);
Ne[i].Contact:= CollectFromDataFile(FileToRead,'[CONTACT]','[/CONTACT]',9,10);
CloseText(FileToRead); }
end else
Ne[i].Status:='NA'; // Ausgabe 'NA' wenn: Datei nicht vorhanden, Status nicht interpretierba
end;
end;
//---------------------------------------------------------------------------
procedure IndexHTML(HTMLDir,ActiveProject:string);
var FileToWrite:text;
begin
```

```
ActiveProject:=ExtractFileNameWithoutExt(ActiveProject);
If FileExists(HTMLDir+ActiveProject+'\index.htm') then
DeleteFile(HTMLDir+ActiveProject+'\index.htm');
MyOpenTextForWrite(FileToWrite,HTMLDir+ActiveProject+'\index.htm');
writeln(FileToWrite,'<html><head><title>index.htm</title></head>');
writeln(FileToWrite,'<frameset rows="15%,85%">');
writeln(FileToWrite,' <frame src="titel.htm" name="oberer Frame" noresize scrolling="no">');
writeln(FileToWrite,' <frameset cols="15%,85%">');
writeln(FileToWrite,' <frame src="status.htm" name="linker Frame" noresize>');
writeln(FileToWrite,' <frame src="bild.htm" name="HauptFrame" noresize>');
writeln(FileToWrite,' <noframes><body> ');
writeln(FileToWrite,' <h1>Alternativ-Inhalt</h1>');
writeln(FileToWrite,'<p>Frames muessen erlaubt sein</p>');
writeln(FileToWrite,' </body></noframes></frameset></frameset></html>');
MyCloseText(FileToWrite);
end;
//---------------------------------------------------------------------------
procedure BildHTML(HTMLDir,VisioDir,ActiveProject:string);
var
Source,Destination : string;
FileToWrite:text;
begin
ActiveProject:=ExtractFileNameWithoutExt(ActiveProject);
Destination:=HTMLDir+ActiveProject+'\gif_1.gif';
if FileExists(VisioDir+ActiveProject+'_files\gif_1.gif')then
Source:=VisioDir+ActiveProject+'_files\gif_1.gif'
else if FileExists(VisioDir+ActiveProject+'-Dateien\gif_1.gif')then
Source:=VisioDir+ActiveProject+'-Dateien\gif_1.gif';
if FileExists(HTMLDir+ActiveProject+'\gif_1.gif')then
DeleteFile(HTMLDir+ActiveProject+'\gif_1.gif');
CopyFile(Source,Destination,true,true); //Quelle,Ziel, RO Flag Quelle mitkopieren?, RO Flag des Zie
If FileExists(HTMLDir+ActiveProject+'\bild.htm') then
DeleteFile(HTMLDir+ActiveProject+'\bild.htm');
MyOpenTextForWrite(FileToWrite,HTMLDir+ActiveProject+'\bild.htm');
writeln(FileToWrite,'<html><head><title>VisioBild</title></head>');
writeln(FileToWrite,'<body width="100%" height="100%">');
writeln(FileToWrite,' <p><img src="gif_1.gif" ></p>');
writeln(FileToWrite,'</body></html>');
MyCloseText(FileToWrite);
end;
//---------------------------------------------------------------------------
procedure TitelHTML(HTMLDir,ActiveProject:string);
var
FileToWrite:text;
begin
ActiveProject:=ExtractFileNameWithoutExt(ActiveProject);
If FileExists(HTMLDir+ActiveProject+'\titel.htm') then
DeleteFile(HTMLDir+ActiveProject+'\titel.htm');
MyOpenTextForWrite(FileToWrite,HTMLDir+ActiveProject+'\titel.htm');
writeln(FileToWrite,'<html><head><title>'+ActiveProject+'</title></head><body>');
writeln(FileToWrite,' <table border="0" cellpadding="5" cellspacing="0" >');
writeln(FileToWrite,' <tr>');
writeln(FileToWrite,' <td><h1>'+ActiveProject+'</h1></td>');
writeln(FileToWrite,' <td></td><td>MOC:</td></tr>');
writeln(FileToWrite,' </table>');
writeln(FileToWrite,'</body></html>');
MyCloseText(FileToWrite);
end;
//---------------------------------------------------------------------------
procedure StatusHTML(HTMLDir,ProjectDir,NeDir,ActiveProject:string );
var
i:integer;
x:string;
FileToWrite:text;
```

```
begin
ActiveProject:=ExtractFileNameWithoutExt(ActiveProject);
If FileExists(HTMLDir+ActiveProject+'\status.htm') then
DeleteFile(HTMLDir+ActiveProject+'\status.htm');
MyOpenTextForWrite(FileToWrite,HTMLDir+ActiveProject+'\status.htm');
writeln(FileToWrite,'<html><head><title>TabellenLayout</title></head>');
writeln(FileToWrite,'<table width="100%" border="2" rules="all">');
writeln(FileToWrite,' <thead><tr><th div align="left">Ne</th><th div align="left">Status</th></tr>
writeln(FileToWrite,' <tfoot><tr><td><i><br></i></td><td><i><br></i></td></tr></tfoot>');
writeln(FileToWrite,' <tbody>');
ActiveProject:=ActiveProject+'.txt';
for i:=0 to ListLength(ProjectDir,ActiveProject)-1 do
begin
If Ne[i].Status = 'OK' then
writeln(FileToWrite,' <tr bgcolor="#00ff00">')
else If Ne[i].Status = 'MINOR' then
writeln(FileToWrite,' <tr bgcolor="#ffff00">')
else If Ne[i].Status = 'MAJOR' then
writeln(FileToWrite,' <tr bgcolor="#ff9900">')
else If Ne[i].Status = 'CRITICAL' then
writeln(FileToWrite,' <tr bgcolor="#ff0000">')
else begin
writeln(FileToWrite,' <tr bgcolor="#999999">');
Ne[i].Status:='NA';
end;
x:=ExtractFileNameWithoutExt(Ne[i].Name);
if FileExists(NeDir+x+'.txt') then
writeln(FileToWrite,' <td>'+x+'</td>') //in zukunft all ergänzung --> link auf statusdate
else
writeln(FileToWrite,' <td>'+x+'</td>');
writeln(FileToWrite,' <td>'+Ne[i].Status+'</td>');
writeln(FileToWrite,' </tr>');
end;
writeln(FileToWrite,' </tbody>');
writeln(FileToWrite,'</table></body></html>');
MyCloseText(FileToWrite);
end;
//---------------------------------------------------------------------------
procedure ProjectOverview(WebProgDir,ProjectDir,CGI_BIN:string);
var
i,j,FileSearch:integer;
FileToWrite:text;
entry:TSearchRec;
list:TStringList;
ActProject:string;
begin
if FileExists(CGI_BIN+'start.htm')then
DeleteFile(CGI_BIN+'start.htm');
MyOpenTextForWrite(FileToWrite,CGI_BIN+'start.htm');
writeln(FileToWrite,'<html><head><title>CurrentProjects</title></head>');
writeln(FileToWrite,'<table width="100%" border="2" rules="all">');
writeln(FileToWrite,' <thead><tr><th></th><th></th><th>Current Projects</th><th></th><th></th></
writeln(FileToWrite,' <tbody>');
writeln(FileToWrite,' <tr>');
list:=TStringList.Create; //immer nötig, bevor man listen verwenden kann
FileSearch:= FindFirst(ProjectDir+'*.txt',faDirectory,entry); //alle projektnamen sammeln
while FileSearch = 0 do
begin
list.Add(entry.name);
FileSearch:=FindNext(entry);
end;
FindClose(entry);
list.Sort;
j:=0;
```

```
try
for i:=0 to list.count-1 do
begin
ActProject:=ExtractFileNameWithoutExt(list.strings[i]);
writeln(FileToWrite,' <th><a href="'+WebProgDir+'Html/'+ActProject+'/index.htm">'+ActProject+'</a>
inc(j);
if j=5 then
begin
writeln(FileToWrite,' </tr>');
writeln(FileToWrite,' <tr>');
j:=0;
end;
end;
finally
list.free;
end;
writeln(FileToWrite,' </tr>');
writeln(FileToWrite,'</table></body></html>');
MyCloseText(FileToWrite);
end;
//---------------------------------------------------------------------------
procedure MakeHtmlFiles(ActiveProject:string); //Erzeugt nötige html files für ein Project
begin
ActiveProject:=ExtractFileNameWithoutExt(ActiveProject);
if not DirectoryExists(IniData.Html+ActiveProject)then
MkDir(IniData.Html+ActiveProject);
//HTML Seiten erstellen
IndexHTML(IniData.Html,ActiveProject);
StatusHTML(IniData.Html,IniData.Project,IniData.Ne,ActiveProject);
TitelHTML(IniData.Html,ActiveProject);
BildHTML(IniData.Html,IniData.XML,ActiveProject);
ProjectOverview(IniData.WebProgDir,IniData.Project,IniData.CGI_BIN);
end;
//---------------------------------------------------------------------------
function FindTestSystemNum(var FileToRead:text; StartKeyword,EndKeyword : string;
StartKeyLength,En
var
DataSearch :string;
position :Integer; //funktion XmlZeileNe ob zeilen in xml, Net
begin
while not eof(FileToRead) do
begin
readln(FileToRead,Datasearch);
position:= pos(StartKeyword,Datasearch);
if position <> 0 then
begin
DataSearch:=copy(DataSearch,position,20);
position:=pos(StartKeyword,DataSearch);
delete(Datasearch,position,StartKeyLength);
DataSearch:=MakePartStr(DataSearch,' ',1);
position:=pos(EndKeyword,DataSearch);
if position <> 0 then
delete(DataSearch,position,EndKeyLength);
position:= pos(' ',Datasearch);
if position <> 0 then
begin
DataSearch:=MakePartStr(DataSearch,' ',1); //Searchstring wird nach leerzeichen a
result:=Datasearch;
end else
begin
DataSearch:=trim(Datasearch);
result:=Datasearch;
end;
end;
```

```
end;
end;
//---------------------------------------------------------------------------
function FindIpsqDir(QuickTIni,VisioHtmlDir,IpsqDir,ActiveProject:string):string;
var
SystemNum,QuickTResult :string;
i,FileSearch,position :integer;
entry :TSearchRec;
FileToRead:text;
begin
ActiveProject:=ExtractFileNameWithoutExt(ActiveProject);
if FileExists(VisioHtmlDir+ActiveProject+'_files\data.xml') then //Englische Visio Version
OpenTextForRead(VisioHtmlDir+ActiveProject+'_files\data.xml',FileToRead,1024) //Pfadangabe + Textda
else
OpenTextForRead(VisioHtmlDir+ActiveProject+'-Dateien\data.xml',FileToRead,1024); //Deutsche Visio V
i:=0;
SystemNum:=FindTestSystemNum(FileToRead,'TA.NE::','</Text>',8,7);
FileSearch:=FindFirst(IpsqDir+'*'+SystemNum,faDirectory,entry);
while FileSearch = 0 do
begin
i:=i+1;
FileSearch:=FindNext(entry);
end;
FindClose(entry);
if i=1 then
begin
//ganzen Verzeichnisnamen zurückgeben
result:=entry.name;
//writeln(result);
CloseText(FileToRead);
end else
begin
//funktion suche in x:\apsk\Tools\QuickT.ini
OpenTextForRead(QuickTIni,FileToRead,1024);
while not eof(FileToRead) do
begin
readln(FileToRead,QuickTResult); //zeile lesen
position:= pos(SystemNum,QuickTResult); //zeiger auf TANummer
if position <> 0 then //wenn TANummer gefunden
begin
position:= pos('.'+SystemNum,QuickTResult);// in Zeile an z.B. ".660" springen
if position <> 0 then
begin
position:= pos('/*',QuickTResult);
if position = 0 then //wenn zeile kein '/*' enthält, dann...
begin
QuickTResult:=CutString(SystemNum+'\',QuickTResult); // löscht alles rechts vo
QuickTResult:=CutStringFromEnd('\',QuickTResult); // löscht alles links von
result:=QuickTResult+SystemNum; //
end;
end;
end;
end;
CloseText(FileToRead);
//writeln(result);
end
end;
//---------------------------------------------------------------------------
Procedure CFToCFGFile(var
CF_Read,CFG_Write:text;ActiveProject,StartKeyword,EndKeyword:string;Start
var
Searchstring:string;
position,line:integer;
begin
```

```
while not eof(CF_Read) do
begin
readln(CF_Read,Searchstring);
position:=pos(StartKeyword,Searchstring);
if position <> 0 then
begin
writeln(CFG_Write,Searchstring);
repeat
readln(CF_Read,Searchstring);
position:=pos(EndKeyword,Searchstring);
line:=pos('{Par97}',Searchstring);
if line <> 0 then
begin
Searchstring := CutString('=',Searchstring);
Searchstring := Searchstring+'=x:\Tools\ipsl\_quicktest\bssap\cases\mmc\mmc';
end;
line:=pos('{Par98}',Searchstring); //Dir Trace File
if line <> 0 then
begin
Searchstring := CutString('=',Searchstring);
Searchstring := Searchstring+'='+IniData.HTML_G+ActiveProject+'\'+ActiveProject+'.trc';
end;
line:=pos('IMSI_A=',Searchstring);
if line <> 0 then
begin
Searchstring:=Copy(Searchstring,1,Length(Searchstring)-3);
Searchstring:=Searchstring+TlnData.IMSI_A;
end;
line:=pos('IMSI_B=',Searchstring);
if line <> 0 then
begin
Searchstring:=Copy(Searchstring,1,Length(Searchstring)-3);
Searchstring:=Searchstring+TlnData.IMSI_B;
end;
line:=pos('{Par96}',Searchstring);
if line <> 0 then
begin
Searchstring := CutString('=',Searchstring);
Searchstring := Searchstring+'= 2';
end;
line:=pos('DUR_A',Searchstring);
if line <> 0 then
begin
Searchstring := CutString('=',Searchstring);
Searchstring := Searchstring+'= '+TlnData.DUR_A;
end;
line:=pos('DUR_B',Searchstring);
if line <> 0 then
begin
Searchstring := CutString('=',Searchstring);
Searchstring := Searchstring+'= '+TlnData.DUR_B;
end;
writeln(CFG_Write,Searchstring);
until (position <> 0);
end;
end;
CloseText(CF_Read);
end;
//---------------------------------------------------------------------
procedure GenerateCFG(VisioHtmlDir,HtmlDir,IpsqDir,ActiveProject:string); //MMC CFG
var
ProjectDir:string;
CFFile,CFGFile:string;
read,write:text;
```

```
begin
ProjectDir:=FindIpsqDir(IniData.QuickTIni,VisioHtmlDir,IpsqDir,ActiveProject); //passendes
if ProjectDir <> '' then
begin
ActiveProject:=ExtractFileNameWithoutExt(ActiveProject);
if FileExists(IPSQDir+ProjectDir+'\ipsq.cf') then // wenn cf file existiert,
begin // dann übergabe Filepfade an read/
CFFile:=IPSQDir+ProjectDir+'\ipsq.cf';
assign(read,CFFile);
CFGFile:=HtmlDir+ActiveProject+'\'+ActiveProject+'.cfg';
assign(write,CFGFile);
end;
If FileExists(HtmlDir+ActiveProject+'\'+ActiveProject+'.cfg') then
DeleteFile(HtmlDir+ActiveProject+'\'+ActiveProject+'.cfg');
OpenTextForRead(CFFile,Read,1024);
MyOpenTextForWrite(write,CFGFile);
CFToCFGFile(Read,Write,ActiveProject,'# CALL04 #','# END CALL04 #',0,0);
CloseText(Read);
MyCloseText(write);
end;
end;
//---------------------------------------------------------------------------
//Hauptprogramm
begin
if paramstr(paramcount)='/debug' then
debug := true else debug := false;
writeln(ExeName,' wird gestartet...');
writeln('');
MyIniDataCollection;
if debug then write('Ftp Dateien von Netzelemente, sammeln...');
ExecuteIt(IniData.NEDATAFTP,false,true,true,true);
if debug then writeln('DONE');
try
if ParamCount < 1 then
begin
Failure:=true;
writeln ('Fehler Parameter');
CreateIOException(255,'Als Parameter die Main.Ini übergeben ',Failure,true);
end;
if debug then write ('IniDaten sammeln...');
if debug then writeln('DONE');
if debug then write ('Ordner leeren...');
DelFiles(IniData.Project+'*.*',true,false); //löscht alle erzeugten ProjectFiles auf X
DelFiles(IniData.Html+'*.*',true,false); //löscht alle erzeugten Html Files auf X
DelFiles(IniData.Html_G+'*.*',true,false); //löscht alle erzeugten Html Files auf G
DelFiles(IniData.ProgDir+'Projekte\*.*',true,false);//löscht alle erzeugten ProjectFiles auf G
DelFiles(IniData.CGI_BIN+'*.*',true,false); //löscht alle erzeugten txt Files im cgi bin
if debug then writeln ('DONE');
if debug then write ('XML Datei lesen...');
ReadXmlFiles(IniData.XML,IniData.Project); // Daten aus Html/XML Sammeln
if debug then writeln ('DONE');
FileSearch:= FindFirst(IniData.Project+'*.txt',faAnyFile,entry);
while FileSearch = 0 do
begin
if debug then writeln('');
if debug then writeln('Daten: '+entry.name);
if debug then writeln('--------------------');
Project:=entry.name;
if debug then write ('Ne Namen sammeln...');
GetNeNames(IniData.Project,Project); // Sammelt Namen der Ne's, übergabe an Re
if debug then writeln('DONE');
if debug then write ('Timestamp pruefen...');
ProofTimestamp(IniData.Project,IniData.Ne,Project); // vergleicht NeDateizeit mit Systemzeit,
if debug then writeln('DONE');
```

```
if debug then write ('Ne Daten sammeln...');
NeDataCollect(IniData.Project,IniData.Ne,Project); // Sammelt Daten aus NeDateien
if debug then writeln('DONE');
if debug then write ('HTML Files erstellen...');
MakeHtmlFiles(Project);
if debug then writeln('DONE');
if debug then write ('CFG File Generieren...');
GenerateCFG(IniData.XML,IniData.Html,IniData.IpsqDir,Project);
if debug then Writeln('DONE');
FileSearch:=FindNext(entry);
end;
FindClose(entry);
if debug then write ('Synchronisation mit Laufwerk G...');
ExecuteIt(IniData.SYNCBAT,false,true,true,true);
if debug then writeln('DONE');
if debug then writeln('Ende');
if debug then Readln;
except
MyExceptHandler;
end;
end.
```

Quellcode TCA

```
{$APPTYPE CONSOLE}
// Parameter 1 = Main.Ini
program tca;
//---------------------------------------------------------------------------
uses
Windows, // THandle, TWIN32FINDDATA, FindFirstFile,...
classes, // TStringList
IniFiles, // TMemIniFile
MyExceptH, // IOExceptions
Sysutils, // FindFirst,FindNext,FindClose,FileExists,CreateDir,Trim,...
Diverses, // MakePartStr, CutString, CutStringFromEnd, ReplaceInString, ExeName, Execute
FilesU, // CreateIOException, ExtractFileNameWithoutExt, OpenTextForRead, CloseText, F
timecnvu, // DatumInSeconds, DOSDateAndTime
FileCtrl, // DirectoryExists
ProcessU; // TYPE,CASE, TSearchRec, TMemIniFile, FileExists, CreateIOException,
//---------------------------------------------------------------------------
Type TIniEntry = record
Project : string;
HTML_G : string;
IPSTool : string;
end;
var
debug :boolean; //
FileRead,FileWrite :text; //lese, schreibvariablen für versch. prozeduren
IniData :TIniEntry;
Project :string;
entry :TSearchRec;
FileSearch :integer;
FileSize :Cardinal;
//---------------------------------------------------------------------------
procedure MyIniDataCollection; //Auslesen aller wichtigen Daten aus der Ini
var
ini : TMemIniFile;
begin
ini:=TMemIniFile.create(ParamStr(1));
try
IniData.Project := ini.ReadString('DESTINATION','PROJECT',''); //Verzeichnis des ProjectOr
IniData.HTML_G := ini.ReadString('DESTINATION','HTML_G','');//
IniData.IPSTool := ini.ReadString('SOURCE','IPSTOOL',''); //Pfad des IPS Tools auf lokaler
finally
```

```
ini.free;
end;
end;
//---------------------------------------------------------------------------
procedure MyOpenTextForWrite(VAR g : Text;Name : string);
begin
if FileExists(Name) then
begin
{$I-}
Assign(g, Name); {Assign --> zuordnen --> Variable "g" bekommt den Namen zugewiesen}
Append(g); {hinzufügen/anfügen von "g"}
CreateIOException(IOResult, 'Append', g, true); {vergleich ob datei schon vorhanden, wenn ja, f
{$I+}
end else begin
{$I-}
Assign(g, Name);
Rewrite(g);
CreateIOException(IOResult, 'Rewrite', g, true);
{$I+}
end;
end;
//---------------------------------------------------------------------------
procedure MyCloseText(VAR g : Text);
begin
{$I-}
case TTextRec(g).Mode of
fmInput, fmOutput, fmInOut:
begin
Close(g);
CreateIOException(IOResult, 'Close', g, true);
end;
end
{$I+}
end;
//---------------------------------------------------------------------------
function CallOK(var FileToRead:text;Keyword: string):boolean;
var
Searchstring :string;
position :Integer;
begin
while not eof(FileToRead) do
begin
readln(FileToRead,Searchstring); //zeile einlesen, übergabe an Datasearch
position:= pos(Keyword,Searchstring); //setze position an konst. string
if position <> 0 then //wenn string vorhanden dann...
begin
Searchstring:=MakePartStr(Searchstring,':',3); //trennt string an jedem ":" und wählt den
SetLength(Searchstring,length(Searchstring)-10); // loescht letzten 10 stellen. uebrig bleib
position:= pos('0',Searchstring); //setzt position an konst string
if position <> 0 then //wenn string vorhanden, dann...
//begin
result:=false
//exit;
else
//begin
result:=true;
//exit;//end;
end;
end;
end;
//---------------------------------------------------------------------------
procedure CaptionHTML(var FileToWrite:text; ActiveProject:string;TestCall:boolean);
begin
ActiveProject:=ExtractFileNameWithoutExt(ActiveProject);
```

```
writeln(FileToWrite,'<html><head><title>'+ActiveProject+'</title></head><body>');
writeln(FileToWrite,' <table border="0" cellpadding="5" cellspacing="0" >');
writeln(FileToWrite,' <tr>');
writeln(FileToWrite,' <td><h1>'+ActiveProject+'</h1></td>');
//writeln(FileToWrite,'<td><input type="button" value="CGI-BIN_Script" onClick="window.navigate('+c
//writeln(FileToWrite,'<td><input type="button" value="Local_Script" onClick="window.navigate('+chr
//writeln(FileToWrite,'<td><input type="button" value="Reset" onClick="window.Location=http://in.bl
if TestCall = true then
begin
writeln(FileToWrite,' <td></td><td bgcolor="#00ff00">MMC:OK</td></tr>');
end else begin
writeln(FileToWrite,' <td></td><td bgcolor="#ff0000">MMC:NOK</td></tr>');
end;
writeln(FileToWrite,' </table>');
writeln(FileToWrite,'</body></html>');
end;
//---------------------------------------------------------------------------
procedure TestCallAnalyzing(var TRCFile,HTMLFile:text; HTMLDir_G,ActiveProject: string);
var
TestCall_MMC:boolean;
begin
if FileExists(HTMLDir_G+ActiveProject+'\'+ActiveProject+'.trc')then
begin
OpenTextForRead(HTMLDir_G+ActiveProject+'\'+ActiveProject+'.trc',TRCFile,1024);
TestCall_MMC := CallOK(TRCFile,'NUMBER OF NOT FAILED CALLS'); //Auslesen trc.file a
CloseText(TRCFile);
if TestCall_MMC=true then
begin
if debug then writeln('Testcall erfolgreich');
If FileExists(HTMLDir_G+ActiveProject+'\titel.htm') then
DeleteFile(HTMLDir_G+ActiveProject+'\titel.htm');
MyOpenTextForWrite(HTMLFile,HTMLDir_G+ActiveProject+'\titel.htm');
CaptionHTML(HTMLFile,ActiveProject,true);
MyCloseText(HTMLFile);
end else begin
if debug then writeln('Testcall nicht erfolgreich');
If FileExists(HTMLDir_G+ActiveProject+'\titel.htm') then
DeleteFile(HTMLDir_G+ActiveProject+'\titel.htm');
MyOpenTextForWrite(HTMLFile,HTMLDir_G+ActiveProject+'\titel.htm');
CaptionHTML(HTMLFile,ActiveProject,false);
MyCloseText(HTMLFile);
end; //Schließen Datei aus Array Liste
end;
end;
//---------------------------------------------------------------------------
function GetFileSize(const szFile: String): Int64;
var
fFile: THandle;
wfd: TWIN32FINDDATA;
begin
result := 0;
if not FileExists(szFile) then exit;
fFile := FindFirstfile(pchar(szFile),wfd);
if fFile = INVALID_HANDLE_VALUE then exit;
result := (wfd.nFileSizeHigh*(MAXDWORD))+wfd.nFileSizeLow;
windows.FindClose(fFile);
//http://www.delphipraxis.net/post191286.html 14.01.2010
end;
//---------------------------------------------------------------------------
//Hauptprogramm
begin
writeln(ExeName,' wird gestartet...');
writeln('');
if paramstr(paramcount)='/debug' then
```

```
debug := true else debug := false;
sleep(30000); //verzögerung wenn enc und tca gleichzeitig gestartet werden
MyIniDataCollection;
FileSearch:= FindFirst(IniData.Project+'*.txt',faAnyFile,entry);
while FileSearch = 0 do
begin
Project:=ExtractFileNameWithoutExt(entry.name);
if FuFileExists(IniData.Html_G+Project+'\'+Project+'.cfg') then //Datei vorhanden ?
begin
FileSize:=GetFileSize(IniData.Html_G+Project+'\'+Project+'.cfg'); //Datei grösser 0 byte?
if FileSize <> 0 then
begin
//x:=IniData.IPSTool;
ExecuteIt(IniData.IPSTool+' '+IniData.Html_G+Project+'\'+Project+'.cfg',false,true,true,tru
//writeln('TestCallTest gestartet...');
TestCallAnalyzing(FileRead,FileWrite,IniData.Html_G,Project);
end;
end;
FileSearch:=FindNext(entry);
end;
FindClose(entry);
if debug then readln;
end.
```

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

Mittweida, 14.03.2012